Mikrokontrolery STM32

Użycie interfejsu I²C, USART, SPI

Wszystkie nieco bardziej zaawansowane systemy mikroprocesorowe muszą komunikować się z układami dodatkowymi. Mogą one być zamontowane na tej samej płytce drukowanej, ale również mogą to być elementy interfejsu zewnętrznego, przykładowo komputer. Wspólnym mianownikiem przy projektowaniu takiego systemu jest wybór odpowiedniego standardu transmisji danych. Wszystkie przedstawione w artykule przykłady zostały uruchomione na płytce ewaluacyjnej STM3210B – EVAL/A, natomiast przedstawione przykłady zostały napisane w oparciu o bibliotekę API dostarczaną przez firmę STMicroelectronics.

Obecnie każdy producent mikrokontrolerów ma w swojej ofercie układy z wbudowaną większością popularnych sprzętowych kontrolerów komunikacyjnych. Znacznie upraszcza to proces tworzenia aplikacji, ponieważ programista nie musi wnikać w szczegóły transmisji. Płytka ewaluacyjna STM3210B – EVAL/A jest wyposażona w mikrokontroler STM32F103VBT, który oferuje w sumie pięć różnych, szeregowych interfejsów komunikacyjnych. Do wykorzystania są: 2×I²C, 3×USART, 2×SPI, CAN, USB 2.0

I²C

Magistrala I²C jest dwukierunkowym, dwuprzewodowym interfejsem komunikacyjnym zaprojektowanym przez firmę Philips. Oryginalnie przeznaczona jest do wymiany informacji pomiędzy układami scalonymi znajdującymi się na tej samej płytce. Na **rys. 1** przedstawiono sposób połączenia dwóch układów magistralą I²C i przykładowy kierunek przesyłania danych oraz sygnału zegarowego. Jak wynika z rysunku, połączenie takie składa się z linii danych (SDA) i linii sygnału zegarowego (SCL).

Sprzętowy kontroler I²C, w jaki wyposażony jest mikrokontroler, umożliwia transmisję z prędkością do 100 kbps w trybie standard, lub do 400 kbps w trybie szybkim. Może pracować jako: podrzędny (*slave*) nadajnik lub odbiornik, nadrzędny (*master*) nadajnik lub odbiornik. Domyślnie kontroler jest ustawiony do pracy jako *slave*.

W celu ukazania zasady nawiązywania komunikacji i wymiany danych uruchomimy prosty przykład. Układ **STM32F103VBT** posiada wbudowane dwa sterowniki magistrali I²C. Można je wykorzystać w przykładzie programowania tak, aby mikrokontroler komunikował się sam ze sobą. Kontroler I2C1 będzie skonfigurowany jako master dla I2C2. Bufor *I2C1_TxBuf[32]*, wypełniony przykładowymi danymi jest wysyłany przez urządzenie I2C1 do urządzenia I2C2. Program realizujący te zadania to plik o nazwie i2c.txt zamieszczony na CD-EP2/2009B. Fizycznie na płytce ewaluacyjnej trzeba połączyć: PB6 (I2C1_SCL) z PB10 (I2C2_SCL) i PB7 (I2C1_SDA) z PB11 (I2C2_SDA). Linie interfejsu (SDA i SCL) muszą być zasilone przez rezystory o wartości 4,7 kΩ.

Kontroler I²C może pracować w jednym z trzech trybów: I²C i dwóch SMBus. Wyboru dokonuje się wypełniając *I2C_Mode* struktury inicjującej. Przedstawiony przykład wykorzystuje I²C, co wskazano przy inicjacji. Ustalenie prędkości transmisji odbywa się przez inicjację pola *I2C_ClockSpeed*. Jeżeli zegar będzie ustalony powyżej 100 kHz, będzie to oznaczać wybór szybkiego trybu komunikacji. Wówczas znaczenia nabiera wartość `I2C_DutyCycle`, która określa relację pomiędzy czasem trwania poziomu wysokiego i niskiego na wyjściu zegarowym. Możliwe do ustawienia relacje to 16 do 9 oraz 2 do 1.

Wartość adresu kontrolera I²C ustala pole `I2C_OwnAddress1`. W zależności od typu adresowania wpisywany jest do niego adres 7- lub 10-bitowy. Ostatnimi czynnościami konfiguracyjnymi jest włączenie lub wyłączenie potwierdzeń.

W ten sposób skonfigurowany, a następnie włączony kontroler I²C jest gotowy do nawiązania komunikacji. Na **rys. 2** przedstawiono przebiegi sygnałów podczas komunikacji I²C. Każdą transmisję rozpoczyna układ nadrzędny wysyłając sekwencję START. Po wykryciu sekwencji startowej wszystkie układy podrzędne przełączają się w tryb odbioru danych, ściślej – adresu. Ostatni bit (ACK) to potwierdzenie odebrania danych, wystawiane przez układ odbierający informację.

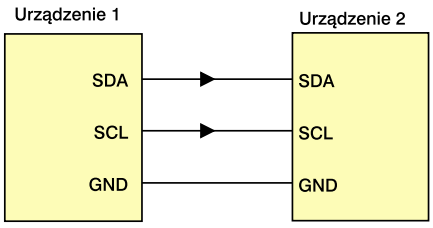
Istotną cechą transmisji I²C jest wymóg stałości sygnału danych podczas trwania stanu wysokiego na linii zegarowej. Jeśli w tym czasie pojawią się stany nieustalone, to przesyłany bit jest nieważny. Na końcu ramki danych transmitowana jest zawsze sekwencja STOP. Jak pokazano na rys. 2, zmiany SDA podczas trwania stanu wysokiego na SCL mogą mieć miejsce tylko przy przesyłaniu znaków START i STOP.

W celu nawiązania komunikacji z odpowiednim układem podrzędnym, należy przesłać jego adres. Ponieważ omawiany przykład nie pracuje w oparciu o przerwania, to należy po każdym poleceniu poczekać na jego wykonanie. Służy do tego funkcja `I2C_CheckEvent()`, która sprawdza, czy podane jako argument zdarzenie miało miejsce i zwraca wartość PRAWDA wtedy, jeśli wynik testu jest pozytywny.

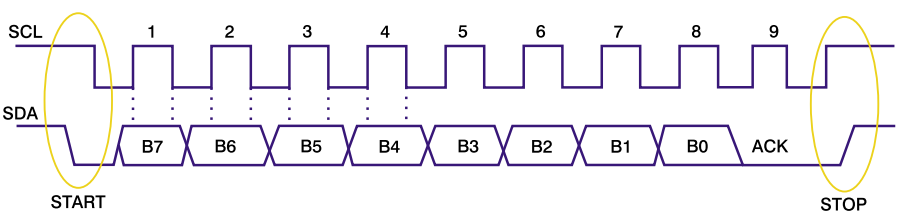
Wysyłanie i odbiór danych wykonuje ta sama pętla `while()`, która wykonywana jest tyle razy, ile wynosi długość bufora wysyłanych danych. Wysyłanie danych do tego samego układu ma raczej niewielki sens i tu służy tylko celom demonstracyjnym.

Uniwersalny synchroniczny/asynchroniczny port szeregowy – USART

Kontroler USART oprócz obsługi standardowej komunikacji szeregowej, może pracować z protokołem IrDA lub SMARTCARD. Pierwszy



Rys. 1. Sposób połączenia układów I²C



Rys. 2. Przebiegi na liniach SDA:SCL podczas transmisji danych

umożliwia kodowanie i dekodowanie danych w standardzie IrDA, co upraszcza aplikacje transmisji danych z użyciem podczerwieni. Do współpracy z portem szeregowym, tak jak w przypadku wszystkich interfejsów komunikacyjnych, może być wykorzystany kontroler DMA.

Komunikacja z terminalem

Uruchomienie omawianego przykładu będzie wymagało podłączenia zestawu ewaluacyjnego do komputera z uruchomionym terminalem portu szeregowego.

W pliku usart.txt zamieszczonym na CD_EP2/2009B umieszczono fragment aplikacji będącej prostą powłoką, umożliwiającą użytkownikowi sterowanie wyjściami mikrokontrolera za pomocą odpowiednich poleceń wydawanych z użyciem terminala. Efektem ich realizacji będzie zmiana stanów wyprowadzeń, co sygnalizuje zaświecanie i gaszenie diod LED. Ponadto naciśnięcie przycisku „Key" powoduje wysłanie przez USART aktualnego stanu diod. Prawidłowe polecenie ma postać: N<nr diody><Enter>. Np. polecenie N6 po naciśnięciu Enter spowoduje pojawienie się stanu wysokiego na wyprowadzeniu PC6 i zaświecenie diody LD1. Analogicznie do gaszenia diod służy polecenie F<nr diody><Enter>. W tym przykładzie będzie to F6 Enter. Sprawdzenie stanu starszej połowy portu C odbywa się przez naciśnięcie przycisku „Key". Jeśli założymy, że zaświecone są diody LD2 i LD4, to zostanie wtedy w terminalu wyświetlony komunikat: onLED: 2,4,

Całość komunikacji obsługiwana jest przez przerwania od kontrolera USART. Parametry transmisji to: prędkość 9600 bps, 1 bit stopu, brak kontroli parzystości.

Po skonfigurowania układu USART do pracy, włączane są przerwania od bufora odbiorczego. Podstawowym zadaniem pętli głównej programu jest sprawdzanie, czy nie został naciśnięty przycisk. Funkcja obsługi przerwań zostanie wywołana za każdym razem, kiedy do bufora odbiorczego portu szeregowego zostanie wpisany nowy bajt. Ponadto, gdy mikrokontroler wykryje stan niski na linii PB9, to po przygotowaniu danych do wysyłki, zostanie włączone przerwanie od bufora nadawczego portu szeregowego. Przerwanie to jest wywoływane za każdym razem, gdy bufor Tx jest pusty.

Odbiór danych

Funkcja obsługi przerwania od portu szeregowego sprawdza, czy przerwanie pochodzi od części nadawczej, czy odbiorczej. Jeśli dane przysyłane są do mikrokontrolera, to ich odczyt z rejestru danych odebranych odbywa się za pomocą funkcji `USART_ReceiveData()`. Zwracana wartość jest zapisywana do tablicy bufora odbiorczego `RxBuf[]`. Teraz funkcja sprawdza, czy ostatni odebrany znak jest kodem CR. Jeżeli tak, to indeks tablicy ustawiany jest na jej początek i ustawiana jest flaga informująca o nadejściu polecenia.

Liczba 1 odejmowana od indeksu tablicy w chwili sprawdzania warunku końca polecenia wynika z użycia postinkrementacji w trakcie odczytu danych z rejestru odbiorczego portu szeregowego. W związku z tym, w momencie sprawdzania końca komunikatu indeks wskazuje na element o jedną pozycję dalej. Po zakończeniu odbioru konieczne jest zdekodowanie łańcucha znaków zawartego w tablicy `RxBuf[]`.

Wszystkie znaki zawarte w tablicy bufora odbiorczego reprezentowane są przez kody ASCII. Spoglądając na tabele kodów ASCII można zaobserwować pewną zależność. Zarówno cyfry jak i litery są uszeregowane w oddzielnych ciągach rosnących. Zmiana kodu ASCII na cyfrę może się odbywać przez zwyczajne odejmowanie od liczby w kodzie ASCII, kodu znaku „zero".

Wyznaczenie właściwego pinu odbywa się dzięki przesuwaniu bitowemu. Zaglądając do pliku nagłówkowego *stm32f10x_gpio.h* z biblioteki API znajdujemy, w jaki sposób są kodowane poszczególne wyprowadzenia. Fragment tego pliku umieszczono poniżej:

```
/* GPIO pins Define ----------------------
-------------*/
#define GPIO_Pin_0 ((u16)0x0001)  /*
Pin 0 selected */
#define GPIO_Pin_1 ((u16)0x0002)  /*
Pin 1 selected */
#define GPIO_Pin_2 ((u16)0x0004)  /*
Pin 2 selected */
#define GPIO_Pin_3 ((u16)0x0008)  /*
Pin 3 selected */
#define GPIO_Pin_4 ((u16)0x0010)  /*
Pin 4 selected */
```

Każdy pin jest reprezentowany przez pojedynczy bit na odpowiadającej mu pozycji. Aby zmodyfikować stan wyprowadzenia należy dokonać operacje przesunięcia bitowego w lewo tyle razy, ile wynosi jego numer. Wszystkie opisane wyżej operacje wykonuje jedna, zwięzła linijka kodu.

Wysyłanie danych

Przerwanie od nadajnika portu szeregowego generowane jest natychmiast po jego włączeniu, jeśli tylko rejestr danych do wysłania jest pusty. Podobnie jak dla odbioru, w pierwszej kolejności należy sprawdzić, czy przerwanie faktycznie pochodzi od nadajnika. Służy do tego funkcja `USART_GetITStatus()`. Następnie, wywołując funkcję `USART_SendData()` i podając jako jej argumenty wywołania numer USART i bajt do wysyłki, wysyłamy porcję danych. Sprawdzanie końca danych przeznaczonych do wysyłki odbywa się w ten sam sposób, jak w przypadku odbioru. Na koniec transmisji wyłączane jest przerwanie od nadajnika USART.

Kodowanie danych do wysłania rozpoczyna się od odczytania stanu portu, do którego są podłączone diody LED (GPIOC). Zajmują one wyprowadzenia PC6…PC9 i dlatego wartość zwracana przez funkcję odczytu stanu portu przesuwana jest bitowo w prawo o 6 pozycji. Pola pozostające z lewej strony dodatkowo są maskowane.

Stały łańcuch, który wysyłany jest do komputera ma długość 7 znaków, więc bufor wysyłki zapisywany jest zaczynając od pozycji ósmej. Sprawdzanie, który bit portu jest ustawiony, a który nie, również wykorzystywane jest za pomocą operacji bitowych. W zależności od tego, który aktualnie cykl jest realizowany przez pętlę `for`, tyle razy wartość zmiennej `stan_portu` przesuwana jest bitowo w prawo. Na koniec maskowane są wszystkie bity oprócz najmłodszego. Jeśli wartość otrzymanego wyrażenia będzie równa 1, to oznacza, że wyprowadzenie mikrokontrolera jest w stanie wysokim. Wówczas wypełniane są dwa kolejne pola tablicy bufora przeznaczonego do wysłania `TxBuf[]`. Diody na płytce ewaluacyjnej są ponumerowane od 1, a nie od 0 i dlatego do końcowej wartości dodawany jest kod cyfry „1". Na ostatniej pozycji wstawiany jest znak CR, a następnie włączana jest przerwanie od nadajnika USART i zawartość bufora wysyłana jest przez port szeregowy do komputera.

SPI

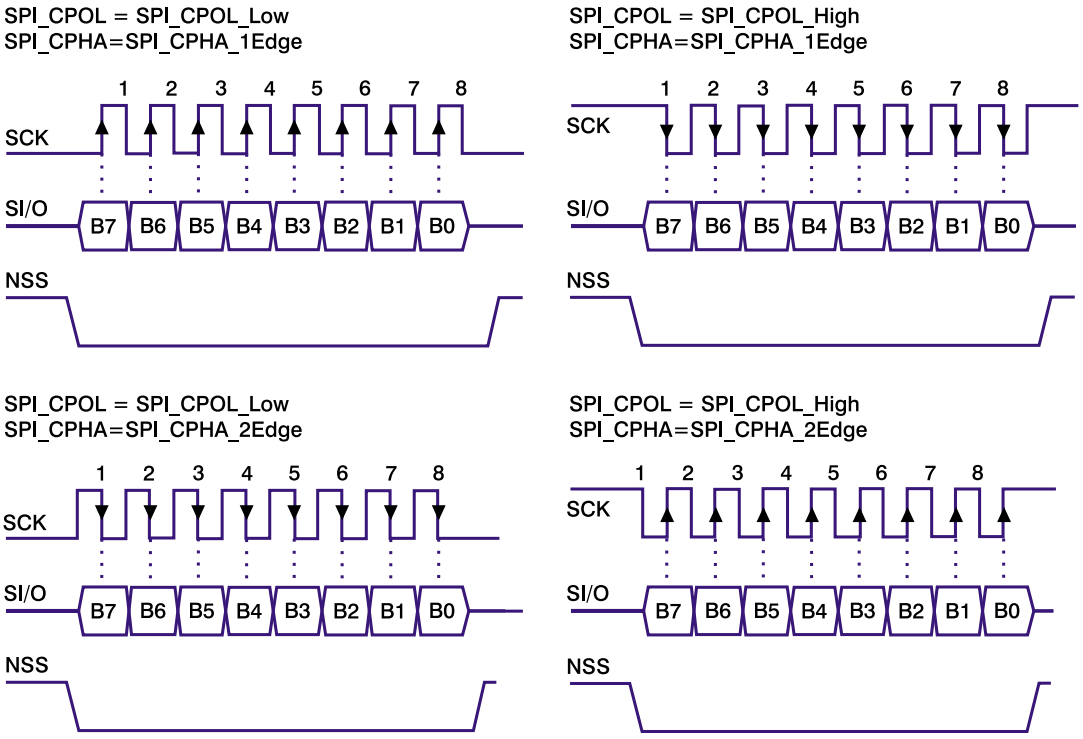
Interfejs SPI (*Serial Peripheral Interface*) służy do dwukierunkowej, synchronicznej transmisji danych. Poprawne nawiązanie komunikacji wymaga trzech linii: zegarowej SCK, MOSI, MISO. Akronim MOSI jest pochodzi od *Master Out/Slave In*, czyli na tej linii dane są przesyłane z układu nadrzędnego (master) do podrzędnego (slave). Analogicznie linia MISO (*Master In/Slave Out*) służy do przesyłania informacji od układu podporządkowanego (slave) do nadrzędnego. Istnieje możliwość skonfigurowania kontrolera SPI do pracy z tylko jedną linią danych.

Kontroler SPI, znajdujący się w mikrokontrolerze STM32F103VBT, umożliwia transmisję z prędkością do 18 Mb/s. Ramka danych może mieć rozmiar 8 lub 16 bitów. Kolejność przesyłania bitów jest konfigurowana.

Podobnie jak to miało miejsce w przypadku I²C, również tutaj zostanie wykorzystana dostępność dwóch kontrolerów sprzętowych. Do uruchomienia przedstawionej aplikacji na płytce ewaluacyjnej należy połączyć: PA5 (SPI1_SCK) z PB13 (SPI2_SCK), PA6 (SPI1_MISO) z PB14 (SPI2_MISO), PA7 (SPI1_MOSI) z PB15 (SPI2_MOSI).

Obydwa kontrolery skonfigurowane są do pracy z pełnym dupleksem. Mikrokontroler musi „wiedzieć", który z interfejsów SPI jest nadrzędny, a który podrzędny. Wyboru dokonuje się poprzez wypełnienie pola `SPI_Mode` struktury inicjującej. Możliwe wartości, jakie może przyjmować to `SPI_Mode_Master` oraz `SPI_Mode_Slave`.

Jak wspomniano wcześniej sprzętowy interfejs SPI w STM32 może pracować z ramkami o długości 8 lub 16 bitów. Przedstawiany przykład wykorzystuje ramki 8-bitowe. Kolejność transmisji bitów jest programowana. Do nastawy służy pole o nazwie `SPI_FirstBit`. Jeśli aplikacja wymaga, aby bity przesyłane były w kolejności od najstarszego do najmłodszego, to polu należy nadać wartość `SPI_FirstBit_MSB`. W przeciwnym przypadku należy użyć `SPI_FirstBit_LSB`.



Rys. 3. Tryby pracy interfejsu SPI

Do wyboru polaryzacji linii zegarowej w stanie spoczynku oraz momentu próbkowania/zmiany stanu linii danych służą odpowiednio nastawy pól `SPI_CPOL` i `SPI_CPHA`.

Nazwa te wynikają wprost z nazw bitów w rejestrze kontrolnym `SPI – SPI_CR1`. Wartość `SPI_CPHA` określa, na którym zboczu sygnału zegarowego stany na liniach danych mają być zatrzaskiwane. W połączeniu z parametrem `SPI_CPOL` otrzymywane są 4 możliwe sytuacje próbkowania linii danych (**rys. 3**).

Kontroler SPI1 skonfigurowano do pracy jako układ nadrzędny, natomiast SPI2 jako układ podrzędny. Transmisję zawsze rozpoczyna i kończy układ nadrzędny (master). Długość ramki wynosi 8 bitów, a częstotliwość linii zegarowej SCK jest równa częstotliwości PCLK podzielonej przez zawartość `SPI_BaudRatePrescaler`. W omawianym przykładzie PCLK jest dzielone przez 2, więc częstotliwość linii zegarowej jest równa około: 36 MHz/128=280 kHz. Jako pierwszy wysyłany/odbierany jest bit najbardziej znaczący.

Nieco obszerniejszego komentarza wymaga pole `SPI_NSS` struktury inicjującej. Odpowiada ono linii NSS wyprowadzonej na zewnątrz mikrokontrolera. Wszystkie urządzenia podłączone do interfejsu SPI mogą być podłączone również do linii NSS. Stan niski NSS może być wymuszony przez układ zewnętrzny i wówczas to on przejmuje kontrolę nad transmisją. Również programowa zmiana roli układu (master/slave) powoduje automatycznie wymuszenie odpowiedniego stanu linii NSS. W omawianym przykładzie funkcja NSS nie jest wykorzystywana.

Dane przesyłane są w pętli `while()`, która wykonuje się tyle razy, ile wynosi rozmiar bufora. Do sprawdzania, czy dany fragment komunikacji został zakończony służy funkcja `SPI_I2S_GetFlagStatus()`. Przy każdym wykonaniu się pętli przesyłany jest jeden bajt z buforów `SPI1_TxBuf[]` i `SPI2_TxBuf[]`. Jeśli rejestr danych przeznaczonych do wysyłki jest pusty, to mikrokontroler rozpoczyna wysyłanie danych przez interfejs SPI1 i SPI2. Następnie, po poprawnym odebraniu przekazywanej informacji, następuje proces wysyłania kolejnej porcji danych.

I²S

Interfejs I²S został stworzony do szeregowej transmisji dźwięku pomiędzy urządzeniami. Jest dostępny tylko w mikrokontrolerach STM32 z najwyższej półki, a więc na płytce STM3210B – EVAL/A nie ma możliwości uruchomienia programów wykorzystujących ten interfejs. Kontroler I²S został tak zaprojektowany, aby jego nastawy odpowiadały standardom dźwięku przesyłanego cyfrowo. Przykładem może tutaj być format danych, do wyboru mamy słowa o długości 16, 24, lub 32 bitów. Pełną specyfikację interfejsu I²S przedstawiono w nocie katalogowej mikrokontrolerów STM32.

Krzysztof Paprocki