Kurs programowania mikrokontrolerów XMEGA (6)

Użycie interfejsu SPI

SPI (ang. Serial Peripheral Interface) jest jednym z trzech najważniejszych interfejsów komunikacyjnych obok I²C oraz UART. W artykule opisano sposób zaprzęgnięcia go do pracy w mikrokontrolerze XMEGA.

Jest to interfejs typu *full-duplex*, a więc umożliwia jednoczesne wysyłanie i odbieranie danych. SPI to interfejs synchroniczny. Jednym z przewodów przesyła się sygnał zegarowy synchronizujący wszystkie układy. Możliwe jest połączenie wielu układów. Najczęściej w sieci mamy jeden układ typu *master*, który wysyła polecenia do układów *slave* i odczytuje z nich dane. Tylko master może rozpocząć transmisję i to on może generować sygnał zegarowy. Sieci SPI multi-master są rzadko spotykane.

Jak działa SPI?

Magistrala SPI składa się z czterech linii. Są to:

- MOSI (master out, slave in) – linia łącząca wyjście danych z mastera i wejścia slave'ów,
- MISO (master in, slave out) – linia łącząca wyjście danych slave i wejście mastera,
- SCK (serial clock) – sygnał zegarowy, synchronizujący układy na magistrali,
- CS (chip select) – sygnał informujący slave o rozpoczęciu transmisji; sygnał ten jest zanegowany, co oznacza się poziomą kreską nad literami CS; uaktywnienie slave'a następuje po wystąpieniu stanu niskiego na linii, a stan wysoki układ slave deaktywuje.

Różni producenci stosują różne nazwy. Zdarzają się skrócone wersje SO i SI lub po prostu O oraz I.

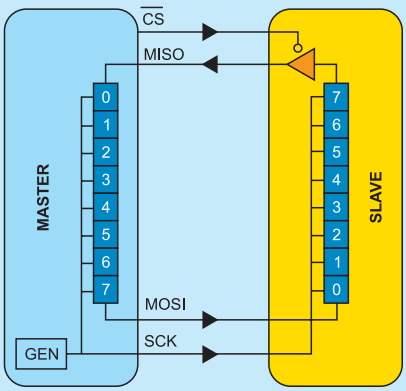
Budowę interfejsu SPI przedstawiono na **rysunku 1**. Jest on oparty na rejestrach przesuwnych. Są to układy składające się z szeregu przerzutników D, synchronizowanych jednym sygnałem zegarowym. W momencie wystąpienia odpowiedniego zbocza zegara, bit zapisany w przerzutniku 1 przesyłany jest do przerzutnika 2, z 2 do 3, z 3 do 4, itd. Do przerzutnika 1 wpisywany jest stan logiczny doprowadzony na wejście łańcucha, natomiast wyjście ostatniego przerzutnika jest wyjściem całego rejestru. Co by było, gdyby wyjście połączyć z wejściem? W takiej sytuacji, dane „kręciły by się w kółko". Konstrukcja taka zwana jest rejestrem pierścieniowym i ma zastosowanie w SPI.

Zarówno master i slave mają rejestr przesuwny, składający się najczęściej z 8 przerzutników, przechowujących 1 bajt danych. Master posiada generator sygnału zegarowego, który steruje pracą wszystkich przerzutników. Linie MISO i MOSI tworzą pierścień, jednak dane pomiędzy masterem i slavem nie kręcą się w kółko w nieskończoność. Co osiem taktów zegara, a więc kiedy zostanie przesłany pełny bajt, zarówno master jak i slave mogą zmienić zawartość swoich rejestrów, przed kolejnym cyklem 8 taktów zegara.

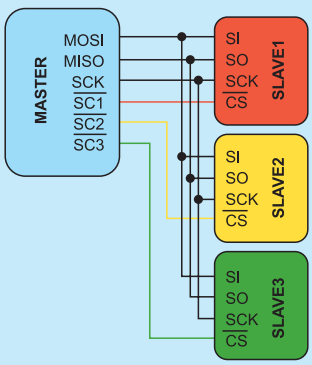
Wynika z tego ważny wniosek – aby master odczytał dane ze slave'a, musi w tym samym czasie coś mu wysyłać. Na ogół są to same zera, jednak należy o tym pamiętać, że SPI jest interfejsem full-duplex.

Wyjaśnienia może wymagać jeszcze bufor trójstanowy na wyjściu ze slave'a. Jest on niezbędny ze względu na to, że do magistrali SPI można podłączyć wiele układów, ale nadawać może tylko jeden z nich. W sytuacji kiedy master dezaktywuje slave'a, bufor trójstanowy odłącza wyjście rejestru przesuwnego od magistrali i slave udaje, że go nie ma. Dzięki temu inny slave może przesyłać dane, bez obawy o konflikt pomiędzy nadajnikami.

Najczęściej spotykaną topologią sieci jest magistrala liniowa (patrz **rysunek 2**). Wszystkie układy są równolegle połączone do linii MISO, MOSI oraz SCK, a więc wszystkie układy „słyszą", co się dzieje na magistrali. Skąd slave'y wiedzą, który z nich ma być odbiornikiem transmisji? Do tego służą oddzielne linie CS, po jednej dla każdego układu slave. W stanie spoczynkowym, na

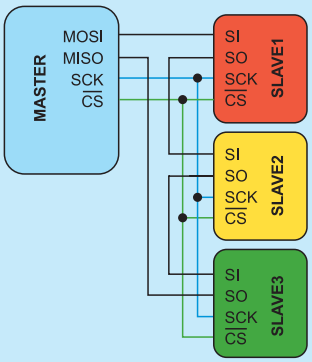


Rysunek 1. Budowa interfejsu SPI



Rysunek 2. Układy SPI połączone w magistralę liniową

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 28637, pass: 752sjb64



Rysunek 3. Układy SPI połączone w łańcuch (daisy chain)

wszystkich liniach CS występuje stan 1 – wtedy wszystkie układy są nieaktywne, a ich wyjścia MISO są ustawione w stan wysokiej impedancji. Master wybiera żądany układ slave ustawiając stan 0 na odpowiedniej linii CS. W sieci o takiej topologii mogą pracować różne układy SPI, niezależnie od ich producenta czy przeznaczenia.

Czasami spotyka się układy SPI połączone w łańcuszek (niektórzy mówią *głuchy telefon* – czasami jest to bardzo trafne określenie takiego połączenia). Wyjście MOSI mastera jest połączone z wejściem slave'a 1, wyjście slave 1 łączy się z wejściem slave 2, itp., a wyjście ostatniego slave łączy się z wejściem MISO mastera. Taką sytuację przedstawia **rysunek 3**. Sygnały SCK oraz CS są połączone do wszystkich slave'ów równolegle.

Takie rozwiązania stosuje się, gdy mamy do czynienia z wieloma układami tego samego typu. Dobrym przykładem są tu sterowniki wyświetlaczy LED oparte na rejestrach przesuwnych. Można ich łączyć dziesiątki lub nawet setki. Innym przykładem, choć nieco odbiegającym od tematyki SPI, jest interfejs JTAG.

SPI nie precyzuje, czy dane mają być wysyłane od najstarszego czy najmłodszego bitu. Możemy sami to ustawić, w zależności od tego, co wymaga konkretny układ slave. Możemy również sprecyzować, czy dane mają być przesuwane w rejestrach po wystąpieniu zbocza rosnącego czy opadającego sygnału zegarowego SCK. Przykładowe konfiguracje i przebiegi sygnałów przedstawiono na **rysunku 4**.

SPI w XMEGA

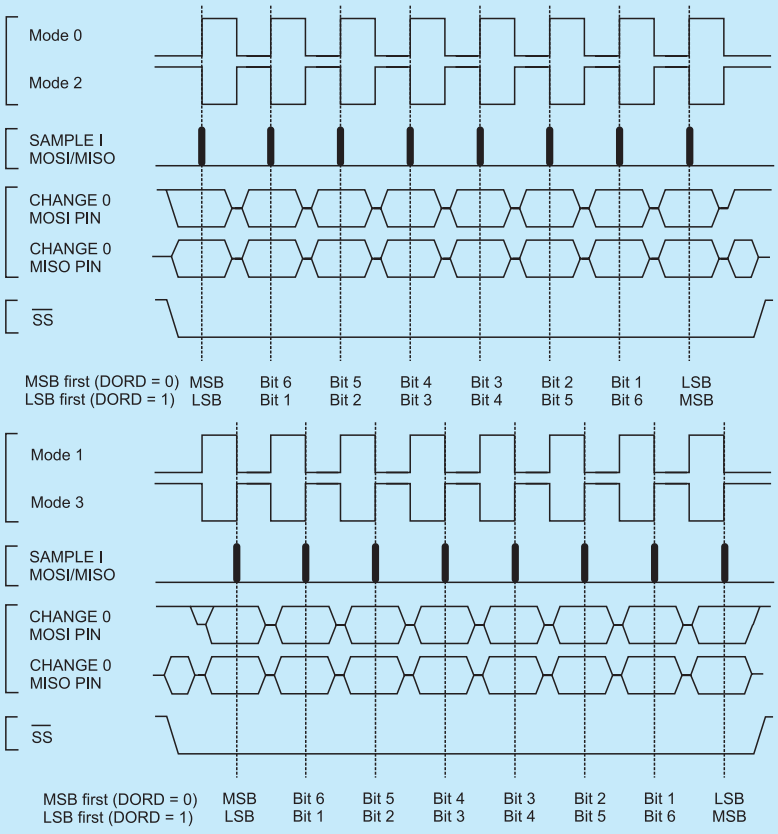
Mikrokontroler ATxmega128A3U posiada trzy interfejsy SPI, dostępne w portach C, D, E, a nazwy tych interfejsów to odpowiednio: SPIC, SPID, SPIE. Są one funkcjonalnie identyczne, a my w naszych przykładach wykorzystamy moduł SPIC. Wszystkie przykłady zostały przedstawione w jednym zbiorczym **listingu 1**.

Pierwsza rzecz, od której najlepiej zacząć, to konfiguracja pinów IO. W starych AVR-ach, włączenie interfejsu powodowało automatyczne skonfigurowanie pinów wejść i wyjść. W XMEGA musimy to zrobić samodzielnie. Konfiguracja pinów IO została opisana w EP 2013/11. Spójrzmy zatem do **ATxmega128A3U datasheet** i w tabeli przedstawionej na **rysunku 5**, sprawdźmy, które piny jaką funkcję realizują. Zamieszczam tą tabelkę, by podkreślić dwie bardzo istotne sprawy:

- Pin SS (slave select) można wykorzystać jako CS do sterowania slavem, kiedy ustawimy go jako wyjście. Jednak jeśli będzie on ustawiony jako wejście, (tak jest domyślnie po włączeniu procesora!), to pojawienie się stanu niskiego na pinie SS spowoduje przełączenie modułu SPI z trybu master do trybu slave. Jeśli Twój procesor zawsze ma pracować w trybie master, ustaw pin SS jako wyjście, nawet jeśli jako CS wykorzystujesz inne piny.
- Przypis 4 pod tabelą mówi, że piny MOSI i SCK można zamienić miejscami. Po co? Zwróć uwagę na interfejs USART C1. Piny MOSI i SCK są funkcjonalnie identyczne jak TXD i XCK. Jeśli projektując płytkę, zamienimy te dwa piny, później pisząc program będziemy mogli wybrać, czy chcemy korzystać z SPI oraz USART. SPI jest prostsze, a USART jest trochę bardziej skomplikowany lecz daje większe możliwości. Zamiany dokonuje się, wpisując wartość **PORT_SPI_bm** do rejestru **PORTx.REMAP**. Pamiętaj, że w płytce eXtrino XL, którą wykorzystamy w tym kursie, piny MOSI i SCK są zamienione.

Po skonfigurowaniu pinów IO, przechodzimy do konfiguracji właściwego interfejsu. Wystarczy wpisać odpowiednie wartości do zaledwie jednego rejestru o nazwie CTRL. Ustawiamy w nim kilka prostych parametrów:

- SPI_ENABLE_bm – ustawienie tego bitu powoduje uaktywnienie interfejsu,
- SPI_MASTER_bm – włączenie trybu master,
- SPI_MODE_x_gc – grupa konfiguracyjna, decydująca m.in. o próbkowaniu i polaryzacji sygnału zegarowego (patrz **rysunek 4**)
- SPI_DORF – przesyłanie danych od najmłodszego bitu,



Rysunek 4. Tryby przesyłania danych

Listing 1. Kod programu demonstrującego działanie SPI w XMEGA

```
#include <avr/io.h>
#include <avr/interrupt.h>

struct PORTX_t {          // struktura danych
    volatile uint8_t IN;  // rejestr wejściowy
    volatile uint8_t OUT; // rejestr wyjściowy
} PORTX;

uint8_t SpiTransmit(uint8_t data) {  // transmisja SPI
    SPIC.DATA = data;  // wysyłanie danych
    while(SPIC.STATUS == 0);  // czekanie na zakończenie transmisji
    return SPIC.DATA;  // odczytanie danych
}

int main(void) {
    // sygnały CS dla peryferiów eXtrino XL
    PORTE.OUTSET = PIN3_bm | PIN6_bm;  // SD, PORTX, DIGPOT
    PORTE.DIRSET = PIN3_bm | PIN6_bm;  // SD, PORTX, DIGPOT
    // konfiguracja SPI
    PORTC.DIRSET = PIN4_bm | PIN5_bm | PIN7_bm; // wyjścia SPI
    PORTC.DIRCLR = PIN6_bm;  // wejście SPI
    PORTC.OUTCLR = PIN7_bm | PIN6_bm | PIN5_bm | PIN4_bm;
    PORTC.REMAP = PORT_SPI_bm;  // zamiana miejscami SCK i MOSI
    SPIC.CTRL = SPI_ENABLE_bm|  // włączenie SPI
                SPI_MASTER_bm|  // tryb master
                SPI_MODE_3_gc|  // tryb 3
                SPI_PRESCALER_DIV64_gc;  // preskaler
    SPIC.INTCTRL = SPI_INTLVL_LO_gc;  // niski priorytet przerwań
    // przerwania
    PMIC.CTRL = PMIC_LOLVLEN_bm;  // włączenie przerwań o priorytecie LO
    sei();
    // pierwsza transmisja
    SPIC.DATA = 0;
    // pętla główna
    while(1) {
        if(PORTX.OUT == PORTX.IN) {  // jeśli wciśnięto przycisk przy świecącej diodzie
            PORTX.OUT = PORTX.OUT << 1;  // przesuń diodę na następną pozycję
            if(PORTX.OUT == 0) PORTX.OUT = 1;  // jeśli ostatnia, zacznij od nowa
        }
    }
}

ISR(SPIC_INT_vect) {
    PORTE.OUTSET = PIN6_bm;  // chip deselect
    asm volatile("nop");  // czekanie na ustabilizowanie się pinu E6
    asm volatile("nop");
    asm volatile("nop");
    asm volatile("nop");
    asm volatile("nop");
    PORTE.OUTCLR = PIN6_bm;  // chip select
    PORTX.IN = SPIC.DATA;  // odczytanie danych
    SPIC.DATA = PORTX.OUT;  // rozpoczęcie nowej transmisji
                            // i wyjście z przerwania
}
```

- SPI_PRESCALER oraz SPI_CLK2X – ustawianie częstotliwości zegara

Jeśli chcemy wykorzystywać przerwania, musimy jeszcze ustawić priorytet przerwań w rejestrze INTCTRL oraz uruchomić kontroler przerwać PMIC (opisane w EP 2013/12). SPI może generować tylko jeden rodzaj przerwań o nazwie **SPIx_INT_vect**.

Do wysyłania i odbierania danych służy rejestr DATA. Pamiętaj, że SPI jest interfejsem full-duplex i nawet jeśli chcesz tylko odebrać jakieś dane, musisz coś wysłać, za przykład 0. Transmisja rozpoczyna się po wpisaniu bajtu danych do rejestru DATA. Następnie należy poczekać, aż dane zostaną przesłane, sprawdzając w pętli rejestr STATUS. Pojawienie się jedynki na pozycji **SPI_IF_bm** oznacza, zakończenie transmisji (co wywoła przerwanie, jeśli jest odblokowane). Dane wysłane ze slave'a do mastera możesz odczytać również z rejestru DATA.

Przed rozpoczęciem transmisji musisz ustawić pin CS wybranego slave'a w stan niski, a następnie poczekać na ustabilizowanie się napięcie na tym pinie. Do tego celu można użyć instrukcji czekania **_delay_us(1)** albo kilkukrotnie wkleić **asm volatile(„nop”);**. Po zakończeniu transmisji, pin CS musisz ustawić w stan wysoki.

Przykłady inicjalizacji, funkcji przesyłającej dane oraz procedury przerwania znajdziesz na **listingu 1**.

Table 32-3. Port C - alternate functions.

PORTC	PIN#	INTERRUPT	TCC0 (1)(2)	AWEXC	TCC1	USART C0 (3)	USART C1	SPIC (4)	TWIC	TWIC w/ext driver	CLOCKOUT (5)	EVENTOUT (6)
PC0	16	SYNC	OC0A	OC0ALS					SDA	SDAIN		
PC1	17	SYNC	OC0B	OC0AHS		XCK0			SCL	SCLIN		
PC2	18	SYNC/ ASYNC	OC0C	OC0BLS		RXD0				SDAOUT		
PC3	19	SYNC	OC0D	OC0BHS		TXD0				SCLOUT		
PC4	20	SYNC		OC0CLS	OC1A			SS				
PC5	21	SYNC		OC0CHS	OC1B		XCK1	MOSI				
PC6	22	SYNC		OC0DLS			RXD1	MISO			RTCOUT	
PC7	23	SYNC		OC0DHS			TXD1	SCK			clk_{PER}	EVOUT
GND	24											
VCC	25											

Notes:
1. Pin mapping of all TC0 can optionally be moved to high nibble of port.
2. If TC0 is configured as TC2 all eight pins can be used for PWM output.
3. Pin mapping of all USART0 can optionally be moved to high nibble of port.
4. Pins MOSI and SCK for all SPI can optionally be swapped.
5. CLKOUT can optionally be moved between port C, D and E and be on pin 4 or 7.
6. EVOUT can optionally be moved between port C, D and E and be on pin 4 or 7.

Rysunek 5. Piny IO w XMEGA

Dodatkowy PORTX

W tym odcinku kursu zobaczymy, jak przy pomocy interfejsu SPI oraz rejestrów przesuwnych typu 74595 oraz 74165 stworzyć dodatkowy port IO oraz jak napisać program, aby obsługa tego dodatkowego portu była podob-

na do zwykłych portów mikrokontrolera. W ten sposób można samodzielnie zbudować odpowiednik ekspandera portu typu PCF8575 za niższą cenę.

Schemat układu przedstawiono **rysunku 6**, a jego uproszczony łatwiejszy do zrozumienia, uproszczony schemat, przedstawia **rysunek 7**. Rejestr 74595 ma wejście szeregowe oraz 8 wyjść równoległych, które możemy wykorzystać do dowolnego celu, np. do sterowania diodami. Układ 74165 ma 8 wejść równoległych i wyjście szeregowe. Możemy do nich podłączyć klawiaturę, czujniki lub inne układy cyfrowe.

Zastanówmy się, co się dzieje w tym układzie podczas transmisji jednego bajtu. Sygnał CS zmienia swój stan z 1 na 0, po czym moduł SPI w XMEGA nadaje osiem bitów. Wraz z każdym taktem sygnały zegarowego SCK, rejestry „przesuwają" swoje bity. W ten sposób, bajt danych z XMEGA trafia do 74595. Paczka 8 bitów dotychczas przechowywanych z 74595 jest przesyłane do 74165 i de facto są to dane, które nas nie interesują. Natomiast 8 bitów z 74165, odpowiadające sygnałom wejściowym tego rejestru, przesyła się do modułu SPI w XMEGA. Na zakończenie, zmiana CS z 0 na 1 powoduje odświeżenie wartości tych rejestrów. W ten elegancki sposób, transmitując 8 bitów, ustaliliśmy stan wyjść oraz odczytaliśmy stan wejść.

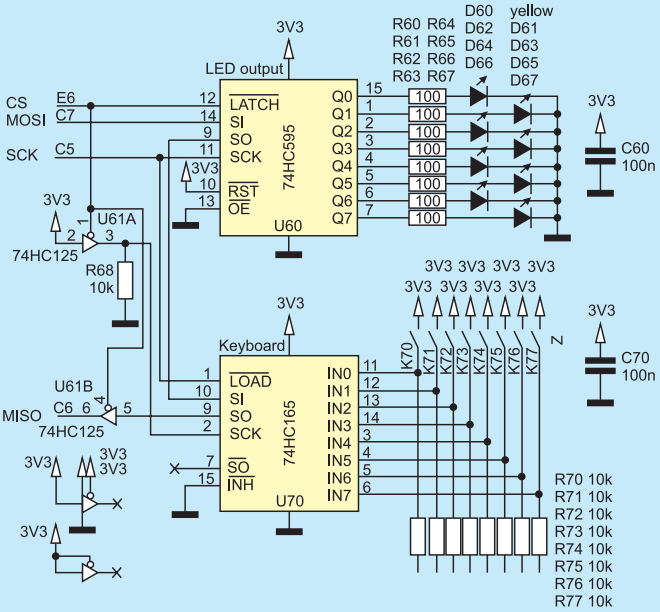
Doskonałym pomysłem jest tutaj zastosowanie przerwań. Po każdej transmisji, układ SPI może generować przerwanie i wysyłać kolejny bajt danych, aby wejścia i wyjścia samoczynnie się odświeżały. Jedyne, co musimy zrobić, to skonfigurować SPI i wywołać pierwszą transmisję, wpisując cokolwiek do rejestru SPIC.DATA.

Możemy posunąć się o krok dalej i stworzyć „pseudoport", by kod programu jak najbardziej przypominał obsługę zwykłego portu. W tym celu zdefiniujemy sobie strukturę PORTX, w której znajdować się będą rejestry IN i OUT, analogicznie do zwykłych portów XMEGA. Ważne, aby nie zapomnieć i kwalifikatorze *volatile*, gdyż te zmienne będą aktualizowane w przerwaniach, a bez tego kompilator mógłby nieprawidłowo je zoptymalizować.

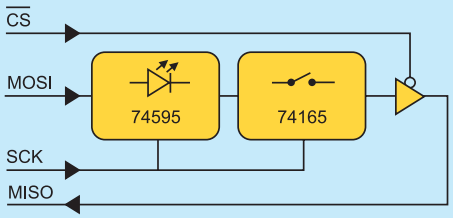
Na płytce eXtrino XL znajduje się 8 przycisków, a przy każdym z nich jest dioda LED. Napiszmy program, w którym świeci się jedna z tych diod tak długo, aż użytkownik naciśnie przycisk przy tej diodzie. Wtedy zapali się sąsiednia dioda i program będzie czekał na naciśnięcie sąsiedniego przycisku, itd.

Kod programu przedstawia **listing 1**. Zwróć uwagę, że w pętli głównej nigdzie nie ma żadnych funkcjo obsługujących SPI. PORTX działa zupełnie jak normalny port!

Wyjaśnić należy procedurę przerwania, gdyż zaczyna się ona od deaktywowania układów SPI (CS=1), następnie czekamy kilka cykli i znów aktywujemy SPI (CS=0). Jest to podyktowane faktem, że ostatnim poleceniem przerwania powinno być wpisanie danych do rejestru SPIC.DATA. Wtedy moduł SPI transmituje dane, a w tym czasie procesor może wykonywać inne zadania. Kiedy



Rysunek 6. Schemat dodatkowego portu sterowanego przez SPI

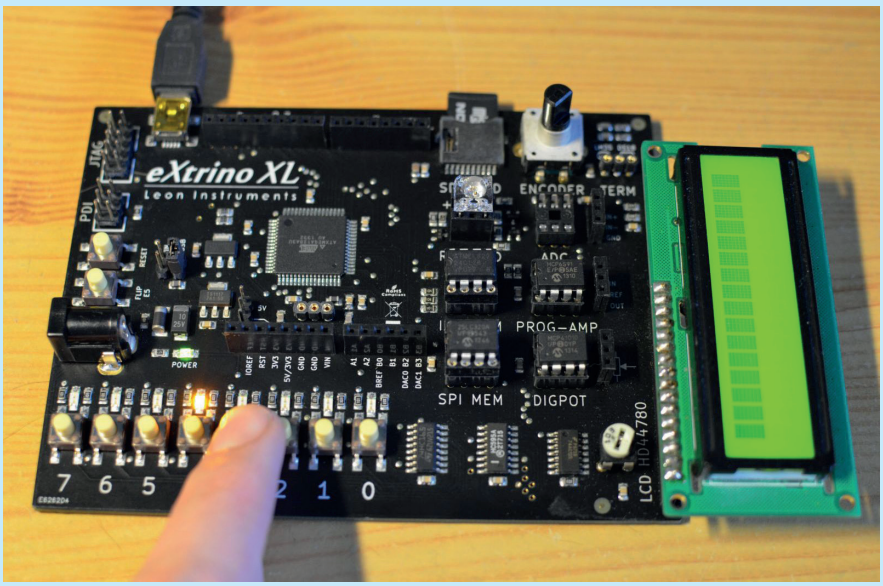


Rysunek 7. Schemat uproszczony

moduł SPI zakończy transmisję, wówczas jest zgłaszane przerwanie. Wtedy, jako pierwsze następuje ustawienie CS w stan wysoki, co oznacza zakończenie transmisji rozpoczętej w poprzednim wywołaniu przerwania. Może to początkowo wydawać się trochę zagmatwane, jednak proszę przeanalizować kod wraz z komentarzami, a wszystko stanie się bardzo proste.

Działanie układu przedstawiono na **fotografii 8**. Po wciśnięciu przycisku, dioda LED „przeskakuje" na sąsiednią pozycję po lewej stronie.

Dominik Leon Bieczyński
www.leon-instruments.pl



Fotografia 8. Przykład działania układu demonstrującego PORTX SPI