

## Podstawy systemów mikroprocesorowych

### Wykład nr 4 Interfejsy szeregowe

dr Piotr Fronczak

<http://www.if.pw.edu.pl/~agatka/psm.html>

[fronczak@if.pw.edu.pl](mailto:fronczak@if.pw.edu.pl)

Pokój 6GF

Komputery przesyłają dane na dwa sposoby:

- równolegle: Kilka bitów danych jest przesyłanych jednocześnie
- szeregowo: Bity przesyłane są jeden po drugim.

Zalety komunikacji szeregowej:

- dłuższe odległości
- łatwiejsza synchronizacja
- mniej kabli
- mniejszy koszt

### Interfejsy szeregowe w mikrokontrolerach AVR

- USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter)
- SPI (Serial Peripheral Interface)
- TWI (2-wire Serial Interface) – odpowiednik I2C
- USB (Universal Serial Bus) – tylko w niektórych
- 1-wire – interfejs jedнопроводowy – brak implementacji sprzętowej

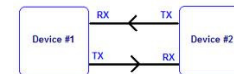
Interfejsy szeregowe umożliwiają łatwe połączenie dwóch urządzeń za pomocą niewielkiej liczby przewodów (2-4).

Łatwa realizacja sprzętowa

Szybkość ograniczona do kilku Mbps

### Interfejs USART

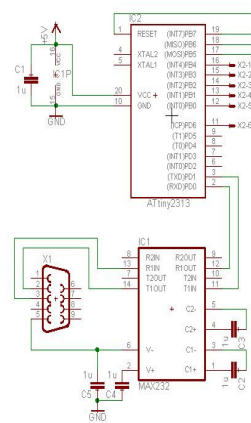
- Dwie linie transmisyjne:
  - › RxD – tor danych odbieranych
  - › TxD – tor danych nadawanych



- Tryby pracy:
  - › Synchroniczny – wykorzystana jest dodatkowa linia taktująca XCK, wykorzystany często w trybie wieloprocessorowym (ang. Multi-processor communication mode)
  - › Asynchroniczny – oszczędność wyprowadzeń mikrokontrolera kosztem większej niepewności transmisji.

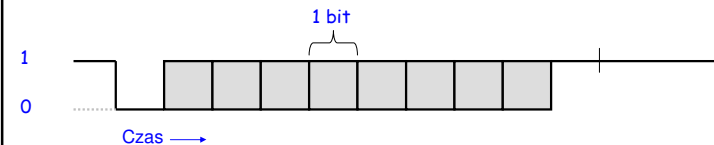
## Interfejs USART – standard RS232

- Najbardziej popularny standard szeregowy - RS232
  - wykorzystywany w komputerach PC
- Standard elektryczny RS232 odbiega od standardu TTL
  - Logiczne 1 – dowolny sygnał od -25V do -3V
  - Logiczne 0 – dowolny sygnał od +3V do 25V
  - Zakres od -3V do +3V nie ma przypisanego poziomu logicznego – detekcja przerwania przewodu



## Interfejs USART – ramka transmisji

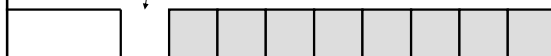
- Jeden przewód do jednokierunkowej transmisji
- Każdy blok reprezentuje jeden bit
- Każdy bit trwa określony czas (konfiguracja szybkości transmisji)
- Przykład: dla prędkości 1200 bps (bitów na sekundę) długość jednego bitu wynosi 1/1200 s (833.3 μs)



## Interfejs USART – ramka transmisji

- Bit startu rozpoczyna transmisję na magistrali RS232.
- Wartość tego bitu zawsze wynosi 0 (informacja dla odbiornika).
- Zbocze opadające jest sygnałem dla odbiornika do synchronizacji odbioru danych

Bit startu



## Interfejs USART – ramka transmisji

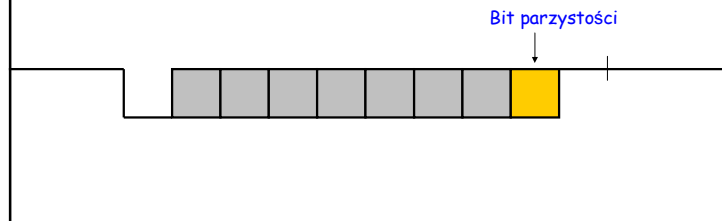
- Po bicie startu wysyłanych jest 5-9 bitów danych
- Najmniej znaczący bit jest wysyłany pierwszy

5-9 bitów danych



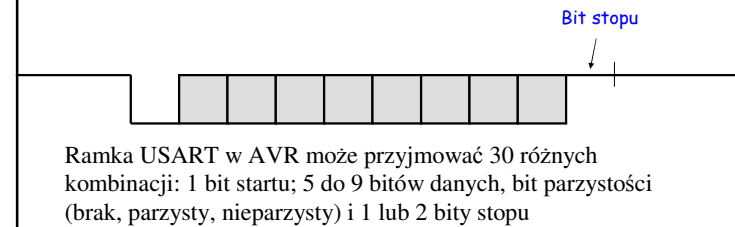
### Interfejs USART – ramka transmisji

- Opcjonalnie wysyłany jest bit parzystości.
- W zależności od konfiguracji interfejsu sprawdzenie, czy całkowita liczba nadanych bitów o wartości 1 była parzysta lub nieparzysta.
- Odbiornik może kontrolować poprawność przesyłanych danych (i ew. sygnalizowany błąd parzystości – *Parity Error*).



### Interfejs USART – ramka transmisji

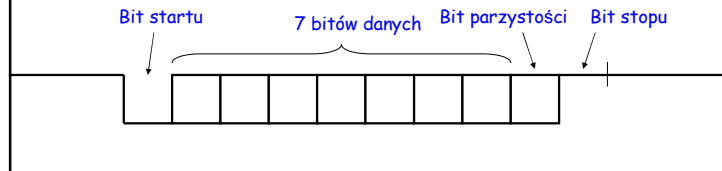
- Transmisję kończy 1-2 bity stopu.
- Bit stopu ma wartość 1.
- Jeśli w tym czasie pojawi się na magistrali jakaś transmisja, odbiornik zinterpretuje ją jako błąd ramki (*Frame Error*).



Ramka USART w AVR może przyjmować 30 różnych kombinacji: 1 bit startu; 5 do 9 bitów danych, bit parzystości (brak, parzysty, nieparzysty) i 1 lub 2 bity stopu

### Interfejs USART – ramka transmisji

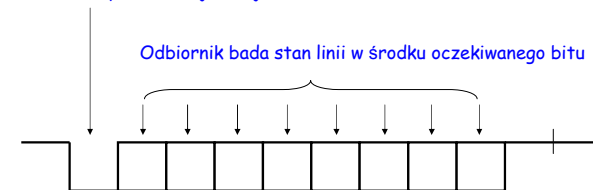
- W poniższym przykładzie, potrzeba 10 bitów do przesłania 7 bitów danych.
- Wydajność transmisji 7/10, czyli 70%
- prędkość transmisji (*baud rate*) != prędkość przesyłania danych (*bit rate*)
- Przykład: Oprócz 8 bitów danych przesyłamy bit startu i stopu. Jeśli 10 bitowa ramka ośmiobitowego słowa ma prędkość transmisji 9600 bodów, to prędkość przesyłania danych wynosi 7680 bps (lub 960 Bps).
- Typowe prędkości: 2400, 4800, 9600, 19200, 38400, 57600, 115200



### Interfejs USART – ramka transmisji

W przypadku interfejsu asynchronicznego istotne jest właściwe taktowanie mikrokontrolera.

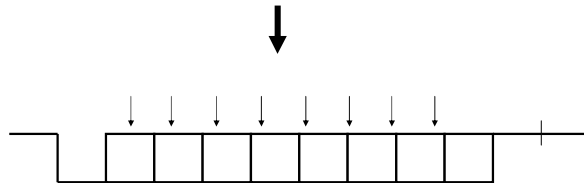
Bit startu informuje odbiornik o rozpoczęciu nadawania,  
Odbiornik synchronizuje swoje liczniki



Odbiornik wykorzystuje swoje liczniki do odmierzenia czasu.  
Szybkość transmisji musi być wcześniej ustalona!

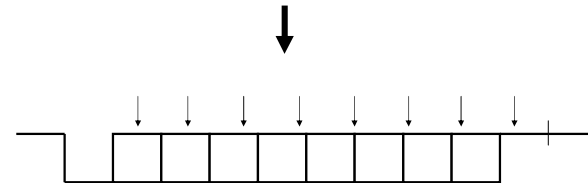
## Interfejs USART – ramka transmisji

Jeśli odbiornik próbkuje sygnał zbyt szybko...

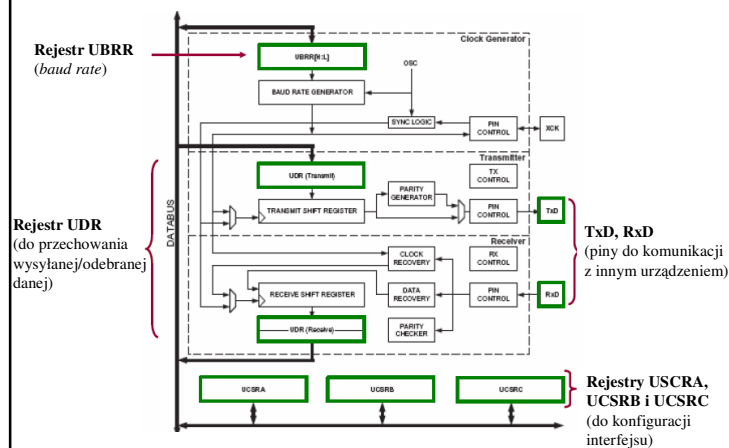


## Interfejs USART – ramka transmisji

Jeśli odbiornik próbkuje sygnał zbyt wolno...

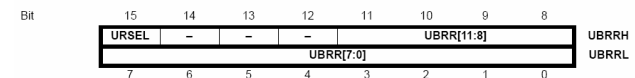


## Interfejs USART w ATmega16



## AVR USART – rejestr UBRR

- Rejestr 16-bitowy (UBRRL, UBRRH)
- W rejestrze UBRR nie podajemy liczby baudów, tylko pewną magiczną liczbę.



Dygresja: Zapisując rejestr UBRRH, bit URSEL musi być = 0.

**Table 74.** Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate <sup>(1)</sup>	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$

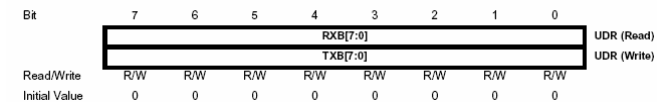
## AVR USART – rejestr UBRR

**Table 82.** Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	$f_{osc} = 1.0000 \text{ MHz}$				$f_{osc} = 1.8432 \text{ MHz}$				$f_{osc} = 2.0000 \text{ MHz}$			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max <sup>(1)</sup>	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

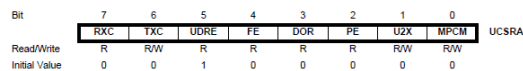
1. UBRR = 0, Error = 0.0%

## AVR USART – rejestr UDR (data register)



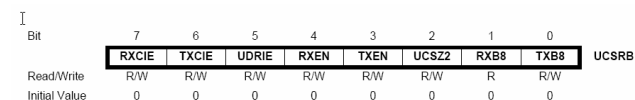
- Bufory transmisji TXB i odbioru RXB dzielą ten sam adres
- Odczytując rejestr UDR odczytujemy zawartość bufora RXB. Zapisując daną do rejestru UDR zapisujemy ją do bufora TXB, co rozpoczyna transmisję.

## AVR USART – rejestr UCSRA



- **RXC** – flaga ustawiana, gdy odebrano daną z magistrali i czeka ona w rejestrze UDR.
- **TXC** – flaga ustawiana, gdy zakończono nadawanie bajtu, a w buforze nadajnika nie ma więcej danych.
- **UDRE** – flaga ustawiana, gdy zawartość rejestru UDR została umieszczona w buforze nadajnika. Można wpisać coś do rejestru UDR.
- **FE** - Framing Error – bit stopu nie ma wartości 1
- **DOR** - Data OverRun – przed zakończeniem odbioru ramki, poprzednią wartość z rejestru UDR należy koniecznie odczytać.
- **PE** - Parity Error - jasne
- **U2X** – podwojenie szybkości transmisji (rzadziej próbkowany stan linii – mogą wystąpić błędy)
- **MPCM** - MultiProcessor Communication Mode - 9 bit ramki = 1 oznacza adres, 0 oznacza daną. Drugi układ, jeśli nie rozpozna swojego adresu nie będzie odbierał danych.

## AVR USART – rejestr UCSRB



- **RXCIE**: włączenie przerwania RXC
- **TXCIE**: włączenie przerwania TXC
- **UDRIE**: włączenie przerwania UDRE
- **RXEN**: Włączenie odbiornika
- **TXEN**: Włączenie nadajnika
- **UCSZ2**: długość słowa

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **RXB8, TXB8**: w trybie 9-bitowym tu znajdują się bity nr 8 przy odbiorze i przy transmisji

## AVR USART – rejestr UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

- URSEL**: wybór rejestru (1=UCSRC / 0=UBRRH)
- UMSEL**: tryb pracy: 0=asynchroniczny, 1=synchroniczny
- UPM1, UPM0**: Parzystość: 00 = wyłączona, 10 = parzysta, 11=nieparzysta
- USBS**: Bity stopu: 0=1 bit, 1= 2 bity
- UCSZ1, 0**: długość słowa

## AVR USART – inicjalizacja

```
void USART_Init( unsigned int baud )
{
    /* prędkość transmisji */
    UBRRH = (unsigned char)(baud>>8);
    UBRRL = (unsigned char)baud;

    /* Format ramki: słowo=8bitów, 2 bity stopu */
    UCSRC = (1<<URSEL) | (1<<USBS) | (3<<UCSZ0);

    /* Włączenie odbiornika i nadajnika */
    UCSRB = (1<<RXEN) | (1<<TXEN);
}
```

## Wysyłanie i odbiór danej (pooling)

```
void USART_Transmit( unsigned char data )
{
    /* Czekaj, aż zwolni się bufor nadajnika */
    while ( !( UCSRA & (1<<UDRE)) );

    /* Umieść daną w buforze i ją wyślij */
    UDR = data;
}

unsigned char USART_Receive( void )
{
    /* Czekaj, aż pojawi się dana do odbioru */
    while ( !(UCSRA & (1<<RXC)) );

    /* Odbierz daną */
    return UDR;
}
```

## Wysyłanie i odbiór danej (przerwania)

```
void USART_Init( unsigned int baud )
{
    /* prędkość transmisji */
    UBRRH = (unsigned char)(baud>>8);
    UBRRL = (unsigned char)baud;

    /* Format ramki: słowo=8bitów, 2 bity stopu */
    UCSRC = (1<<URSEL) | (1<<USBS) | (3<<UCSZ0);

    /* Włączenie odbiornika i nadajnika */
    UCSRB = (1<<RXEN) | (1<<TXEN);

    /* Włączenie przerwania, gdy przyszła dana */
    UCSRB |= (1<<RXCIF);

    /* Włączenie przerwania, gdy bufor nadawczy pusty */
    UCSRB |= (1<<UDRIE);
}
```

## Wysyłanie i odbiór danych (przerwania)

- przerwanie (USART\_RXC\_vect) będzie wywoływane tak długo, jak będą nieodebrane dane w rejestrze UDR
- procedura przerwania MUSI odczytać daną lub wyłączyć możliwość przerw

```
ISR (USART_RXC_vect) {
    static int rxindex=0;
    RXBuf[rxindex++] = UDR;
}
```

- przerwanie (USART\_UDRE\_vect) będzie wywoływane tak długo, jak UDR będzie pusty
- procedura przerwania MUSI zapisać nową daną do UDR albo wyłączyć możliwość przerw

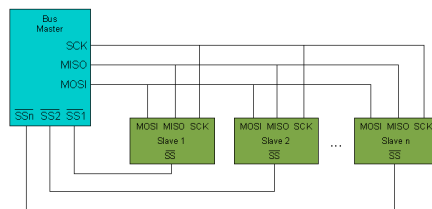
```
ISR (USART_UDRE_vect) {
    static int txindex=0;

    if (TXBuf[txindex]) // jeśli są znaki w buforze
        UDR=TXBuf[txindex++];
    else { txindex=0; UCSRB &= ~(1<<UDRIE); }
}
```

## Interfejs SPI

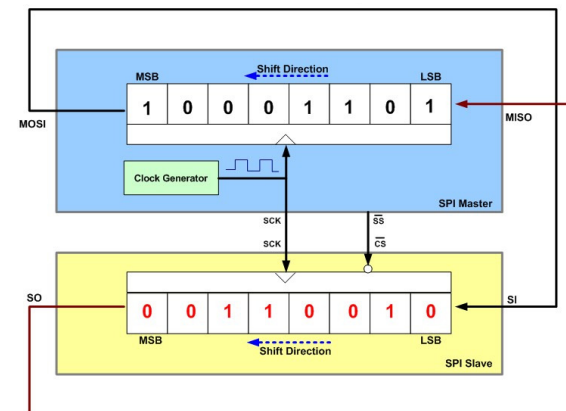
- SPI – *Serial Peripheral Interface*
- Opracowany przez firmę Motorola
- Interfejs synchroniczny, czteroprzewodowy
- Transmisja typu duplex
- Dwa rodzaje urządzeń na magistrali:
  - Układ *Master* – steruje przepływem danych na magistral
  - Układ *Slave* – układ wysyłający i odbierający dane
- Gwarantowana prędkość transmisji: 2,1Mbd (niekiedy nawet do 10Mbd)

## Podłączanie urządzeń do magistrali, linie interfejsu



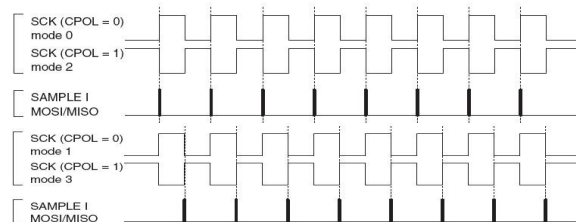
- Linia sygnału zegarowego (*Serial Clock*) SCK
- Wyjście danych układu *Master* (*Master Out Slave In*) MOSI
- Wejście danych układu *Master* (*Master In Slave Out*) MISO
- Wybór układu *Slave* (*Slave Select*) - SS

Dane są jednocześnie wysyłane i odbierane (*full-duplex*)



## Tryby pracy magistrali SPI

- 2 parametry konfiguracyjne: polaryzacja (*Clock Polarity* - CPOL) i faza (*Clock Phase* - CPHA) sygnału zegarowego, określają sposób inicjowania transmisji zboczem zegara
- W przypadku sprzętowego interfejsu SPI, najczęściej jest możliwość wyboru trybu transmisji.
- Układy *Slave* mają najczęściej na stałe ustalony tryb pracy (powszechnie używanie – 1 i 3)



## Rejestr SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- SPIE: włączenie przerwań SPI (gdy transmisja zakończona)
- SPE: włączenie obsługi interfejsu
- DORD: 1 - LSB wysłany pierwszy / 0 - MSB wysłany pierwszy
- MSTR: 1 - tryb Master, 0 - tryb Slave.
- CPOL: polaryzacja, 0 (1) – linia SCK w stanie bezczynności ma poziom niski (wysoki)
- CPHA: faza zegara – 0 (1) – próbkowanie na pierwszym (drugim) zboczach zegara
- SPR1:0 : częstotliwość próbkowania

SPR2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

## Rejestr SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- SPIF: Flaga przerwania SPI
- WCOL: flaga kolizji (gdy rejestr danych SPDR jest nadpisany w trakcie transmisji)
- SPI2X: podwojenie szybkości transmisji

## Rejestr SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Wpisanie danej do SPDR rozpoczyna transmisję

## Inicjalizacja modułu SPI w trybie Master

```
#define MOSI PB5
#define SCK PB7
#define SS PB4

void InitSPI (void)
{
    //ustawienie kierunku wyjściowego dla linii MOSI, SCK i SS
    DDRB |= (1<<MOSI)|(1<<SCK)|(1<<SS);

    //aktywacja SPI, tryb Master, prędkość zegara Fosc/64
    SPCR |= (1<<SPE)|(1<<MSTR)|(1<<SPR1);
}
```

Uwaga: sprawdzić w notcie katalogowej, których linii kierunek należy ustawić, a których jest automatycznie wymuszany.



## Odbiór i wysłanie bajtu

```
uint8_t TransferSPI(uint8_t bajt)
{
    SPDR = bajt;
    //czekamy na ustawienie flagi SPIF po zakończeniu transmisji
    while( !(SPSR&(1<<SPIF)) );
    return SPDR;
}
```

## Inicjalizacja modułu SPI w trybie Slave

```
void InitSPISlave (void)
{
    //ustawienie kierunku wyjściowego dla linii MISO
    DDRB |= (1<<MISO);
    //aktywacja SPI, tryb Slave
    SPCR |= (1<<SPE);
}
```

## Przykład: ekspander portów

Wykorzystamy 8 bitowy rejestr przesuwny z zatraskami do rozszerzenia portów w mikrokontrolerze. Układ tego typu pozwala na szeregowe przesyłanie danych do własnego rejestru, a następnie wyprowadzenie ich na zewnątrz jednocześnie do tzw. zatrasku.

<http://www.ermicro.com/blog>

<http://www.circuitworld.com.br>

Top View

Truth Table

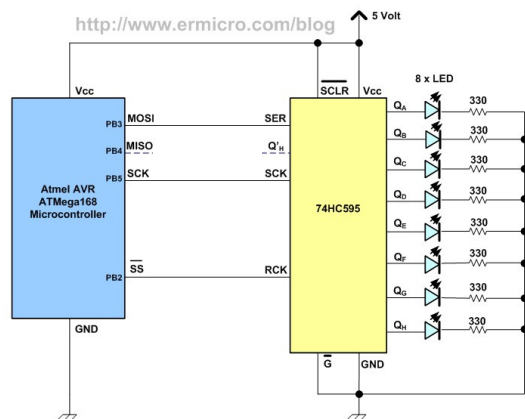
RCK	SCK	SCLR	$\overline{G}$	Function
X	X	X	H	$Q_A$ thru $Q_H = 3\text{-STATE}$
X	X	L	L	Shift Register cleared $Q_H = 0$
X	$\uparrow$	H	L	Shift Register clocked $Q_n = Q_{n-1}$ , $Q_0 = \text{SER}$
$\uparrow$	X	H	L	Contents of Shift Register transferred to output latches

H – Logical High, L – Logical Low, X - Don't Care

$\uparrow$  - Rising Clock Pulse



74HC595 8-Bit Shift Registers with Output Latches



```
#include <avr/io.h>
#include <util/delay.h>
#define MOSI PB5
#define SCK PB7
#define SS PB4

void SendSPI(uint8_t bajt) {
    SPDR = bajt; //wysyłamy bajt do układu Slave
    while( !SPSR & (1<<SPIF)); // czekamy na zakończenie trans.
    PORTB |= (1<<SS); // zbocze narastające zatraskuje wartości
    // rejestru do wyjść Q<sub>A</sub>-Q<sub>H</sub>
    _delay_us(1); // czekamy
    PORTB &= ~(1<<SS); // przywracamy stan niski na linii zatrasku
}

int main(void) {
    InitSPI(); // inicjalizacja SPI - zrobiliśmy to wcześniej
    while(1) {
        cnt=1;
        while(cnt) {
            SendSPI( cnt );
            _delay_ms(100);
            cnt <<= 1;
        }
    }
}
```