Co-funded by the
Erasmus+ Programme
of the European Union

"Teaching online electronics, microcontrollers and programming in Higher Education"
Erasmus+ Programme
2020-1-PL01-KA226-HE-095653

Podstawy systemów mikroprocesorowych: Interfejsy szeregowe

dr inż. Dariusz Tefelski

dariusz.tefelski@pw.edu.pl

Pokój 225GF

- Mikrokontroler ATmega32 ma magistralę 8-bitową. Zatem naturalnym wyborem jest przesyłanie informacji poprzez jednoczesne przesłanie 8-bitów.
- Aby przesłać 8-bitów jednocześnie potrzeba 8 wyprowadzeń.
- Istnieje możliwość przesłania danych korzystając tylko z jednego wyprowadzenia, dzieląc bajt danych bit po bicie.

- Zatem wyróżniamy dwa sposoby przesyłania danych:
  - równoległy - kilka bitów danych jest przesyłanych jednocześnie,
  - szeregowy - bity przesyłane są jeden po drugim.

Porównanie komunikacji równoległej i szeregowej

Zalety komunikacji równoległej:

- łatwa w obsłudze.

Wady komunikacji równoległej:

- wymaga wielu wyprowadzeń.

Zalety komunikacji szeregowej:

- dłuższe odległości,
- mniej przewodów,
- mniejszy koszt.

Wady komunikacji szeregowej:

- zazwyczaj większa komplikacja układów nadajnika i odbiornika aby poprawie zdekodować informacje,
- proste interfejsy (np. RS232) są zwykle powolne.

Jednak pomimo wad, komunikacja szeregowa jest szeroko stosowana np. interfejs HDMI, połączenia do dysków Serial ATA czy interfejs PCI Express w płytach głównych komputerów.

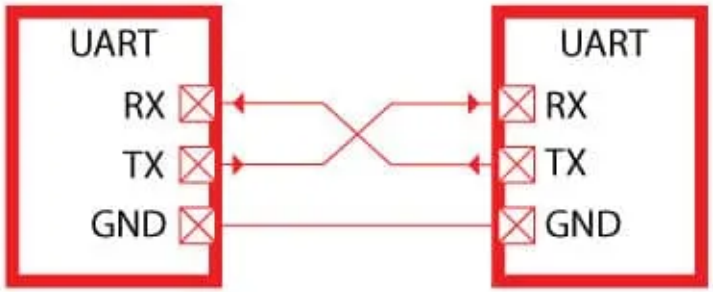
Interfejsy szeregowe w mikrokontrolerach AVR

- USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter) - wykorzystywany do asynchronicznego przesyłania danych np. w RS232 oraz do synchronicznej komunikacji z innymi mikrokontrolerami,
- SPI (Serial Peripherial Interface) - zwykle stosowany do komunikacji z innymi peryferiami zewnętrznymi/układami scalonymi, kartami SD oraz do programowania mikrokontrolerów z rodziny AVR.
- TWI (2-wire Serial Interface) - odpowiednik I2C. Ze względu na patentowe ograniczenia został nazwany TWI. Tak samo jak SPI jest interfejsem synchronicznym ale o mniejszej liczbie połączeń.
- USB (Universal Serial Bus) - tylko w niektórych mikrokontrolerach AVR.
- 1-wire - intefejs jednoprzewodowy, brak implementacji sprzętowej ale łatwo można go zaimplementować programowo.

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.

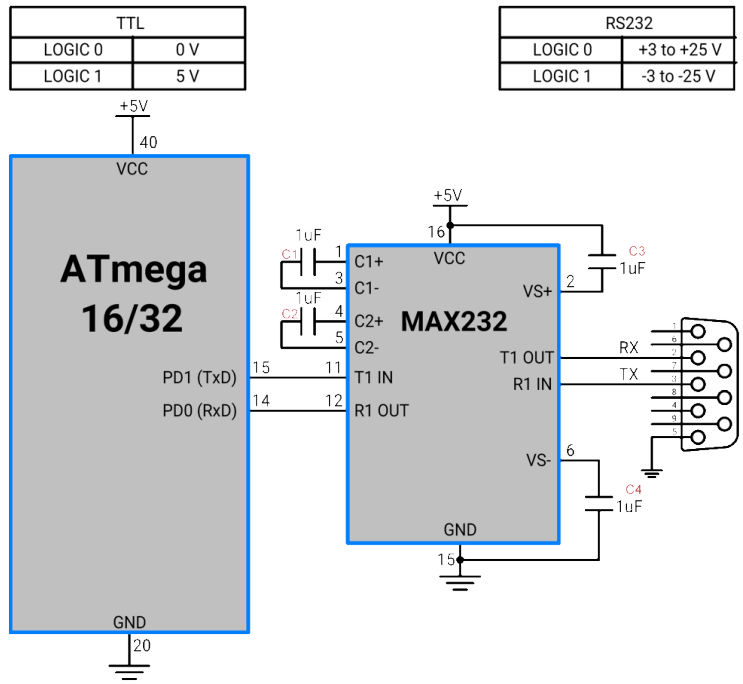


Source: `https://microcontrollerslab.com/uart-communication-working-applications/`

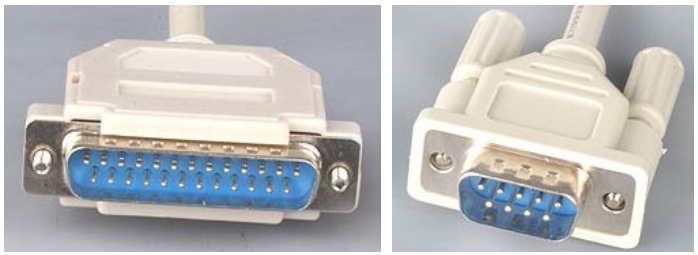
- Tryby pracy:
  - Synchroniczny - wykorzystana jest dodatkowa linia taktująca XCK, wykorzystany często w trybie wieloprocesorowy (*Multi-processor communication mode*) np. do komunikacji pomiędzy kilkoma mikrokontrolerami.
  - Asynchroniczny - oszczędność wyprowadzeń mikrokontrolera (RxD, TxD oraz GND) kosztem większej niepewności transmisji. Stosowany np. do komunikacji z komputerem.

Interfejs USART - standard RS232

- Najbardziej popularny standard szeregowy - RS232
  - wykorzystywany w komputerach PC. Obecnie rzadko jest złącze RS232 w komputerach ale stosuje się przejściówki.
- Standard elektryczny RS232 odbiega od standardu TTL (wykorzystywanego w mikrokontrolerach)
  - 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 - służy np. do detekcji przerwanego przewodu.



Source: https://www.electronicwings.com/avr-atmega/atmega1632-usart

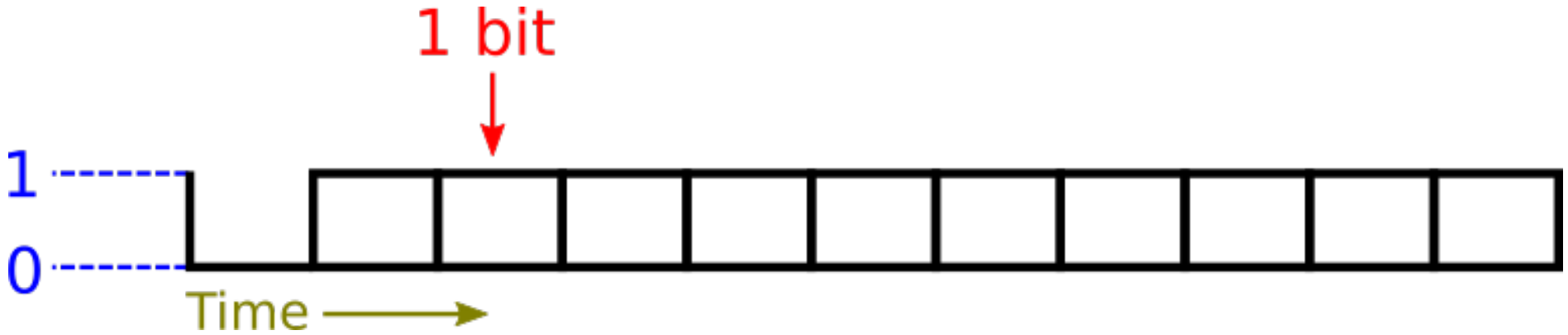


Source: https://www.meditronik.com.pl/

- Większe różnice między stanami logicznymi sprawiają że interfejs jest bardziej odporny na zakłócenia ale nie tak bardzo jak interfejsy posługujące się sygnałami prądowymi np. RS485.

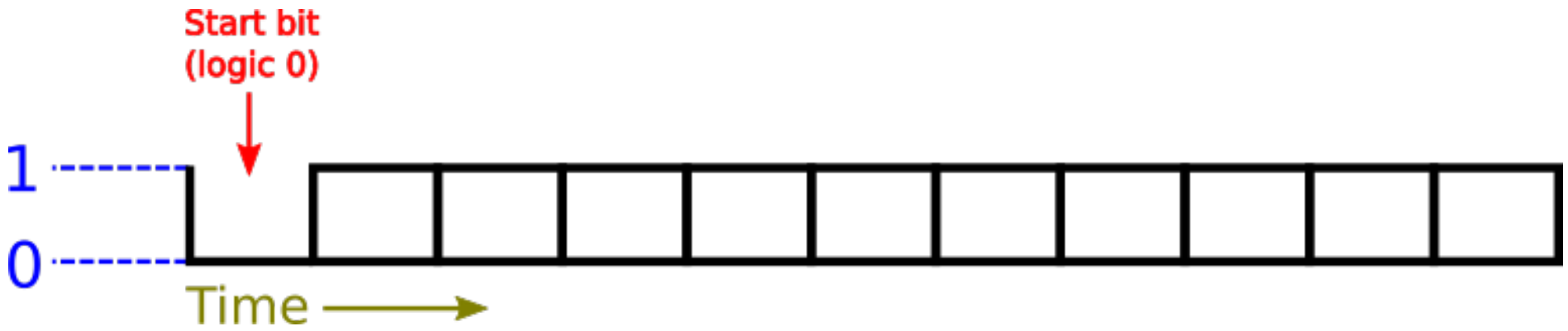
Interfejs USART - ramka transmisji

- Jeden przewód do jednokierunkowej transmisji.
- Dane są transmitowane w postaci ramki zawierającej sekwencję bitów.
- 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).



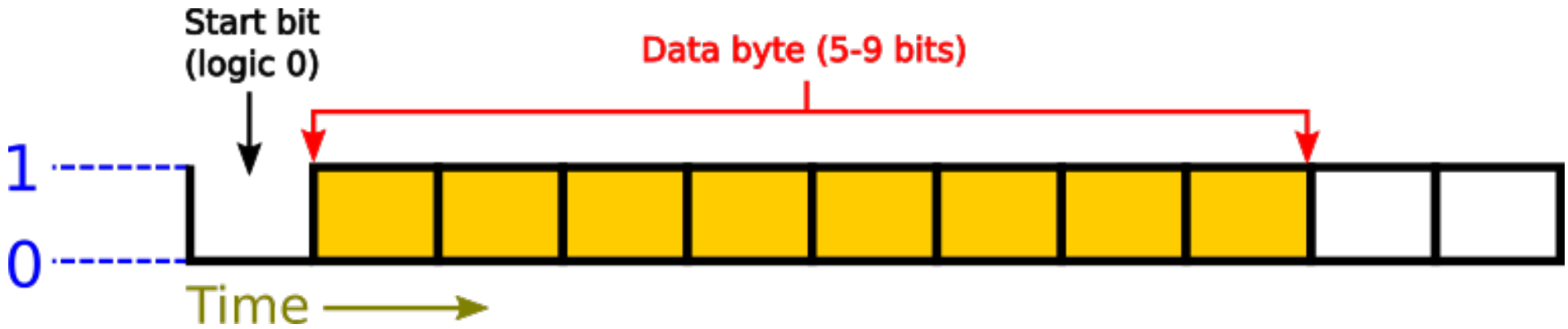
Interfejs USART - ramka transmisji

- 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.



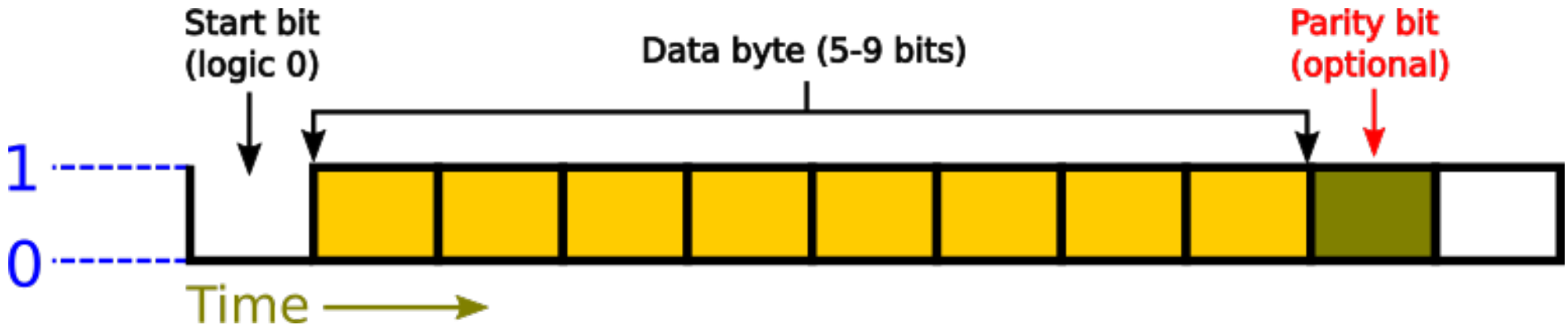
Interfejs USART - ramka transmisji

- Po bicie startu wysyłanych jest 5-9 bitów danych.
- Zazwyczaj wykorzystuje się 8 bitów danych gdyż mikrokontroler jest 8 bitowy.
- Natomiast 7 bitów danych wystarczy do przesyłania znaków w kodzie ASCII.
- Najmniej znaczący bit jest wysyłany pierwszy a potem kolejne aż do najbardziej znaczącego bitu.



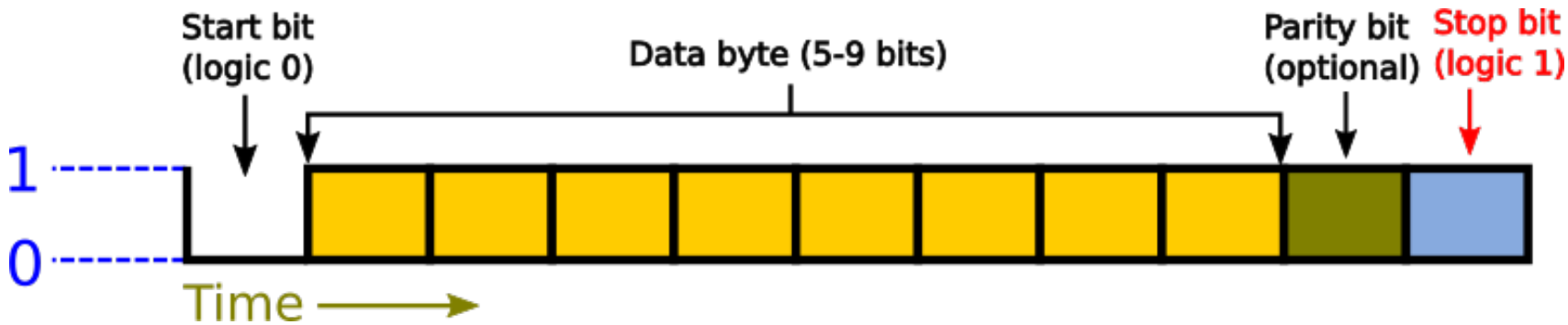
Interfejs USART - ramka transmisji

- 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

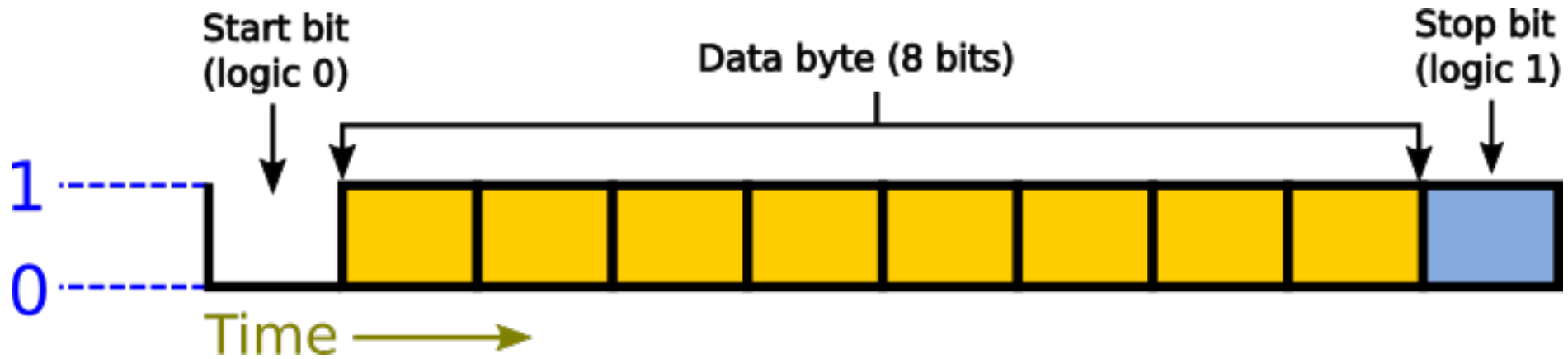
- Transmisję kończy 1-2 bity stopu.
- Bit stopu ma wartość 1.
- Jeśli w tym czasie pojawi się na magistrali jakaś transmisja, odbiornik zinterpretuje ją jako błąd ramki (*Frame Error*).



- 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.

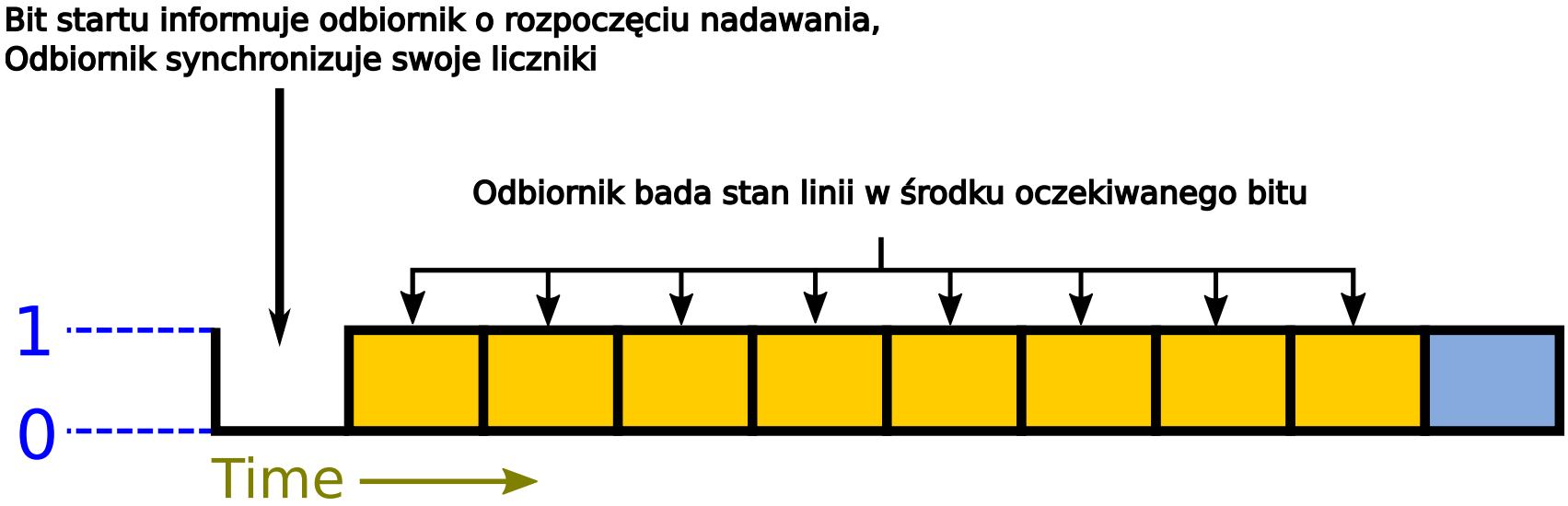
Interfejs USART - ramka transmisji

- W poniższym przykładzie, potrzeba 10 bitów do przesłania 8 bitów danych.
- Wydajność transmisji 8/10, czyli 80%.
- Prędkość transmisji (*baud rate*) != prędkość przesyłania danych (*bit rate*).
- **Przykład:** Jeśli 10 bitowa ramka 8-bitowego słowa ma prędkość transmisji 9600 bodów, to prędkość przesyłania danych wynosi: 960 Bps (Baud rate/długość ramki) lub 7680 bps (prędkość w Bps*liczba bitów danych).
- Bps - *Bytes per second*, bps - *Bits per second*.
- Typowe prędkości (baud rate): 2400, 4800, 9600, 19200, 38400, 57600, 115200.
- W mikrokontrolerze stosuje się preskalery aby uzyskać pożądane prędkości.



Interfejs USART - ramka transmisji

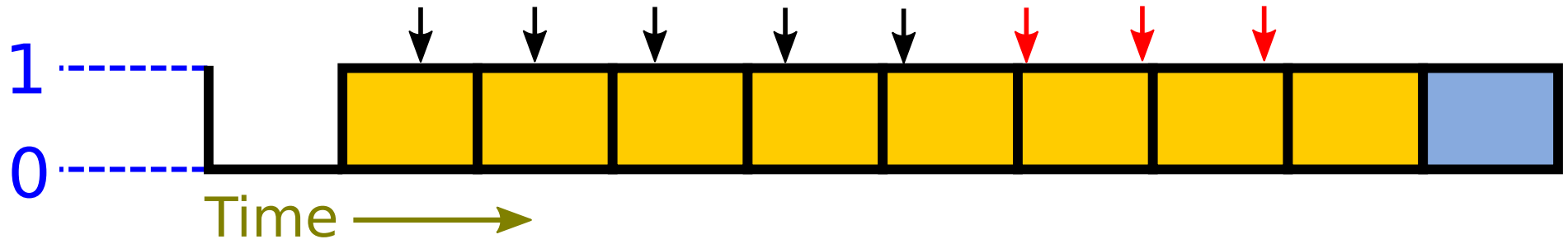
- W przypadku interfejsu asynchronicznego istotne jest właściwe taktowanie mikrokontrolera.



- W rzeczywistości, mikrokontroler próbkuje stan każdego bitu kilka razy i w ten sposób może poprawić jakość odczytu.

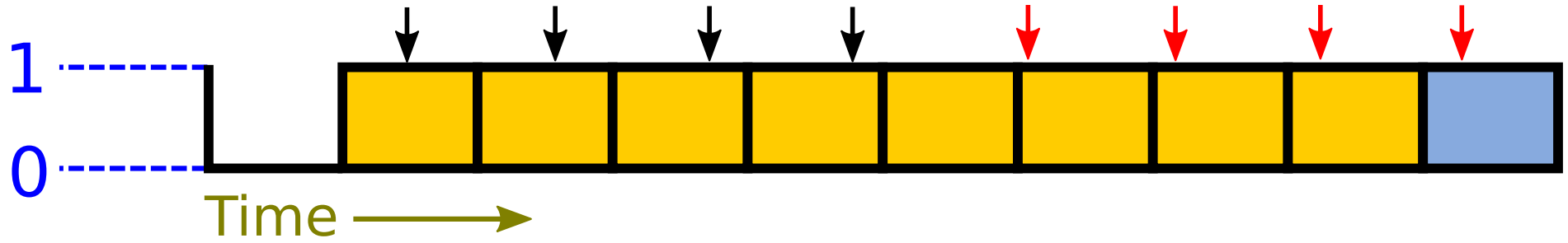
Interfejs USART - ramka transmisji

Co się dzieje gdy odbiornik próbkuje sygnał zbyt szybko:



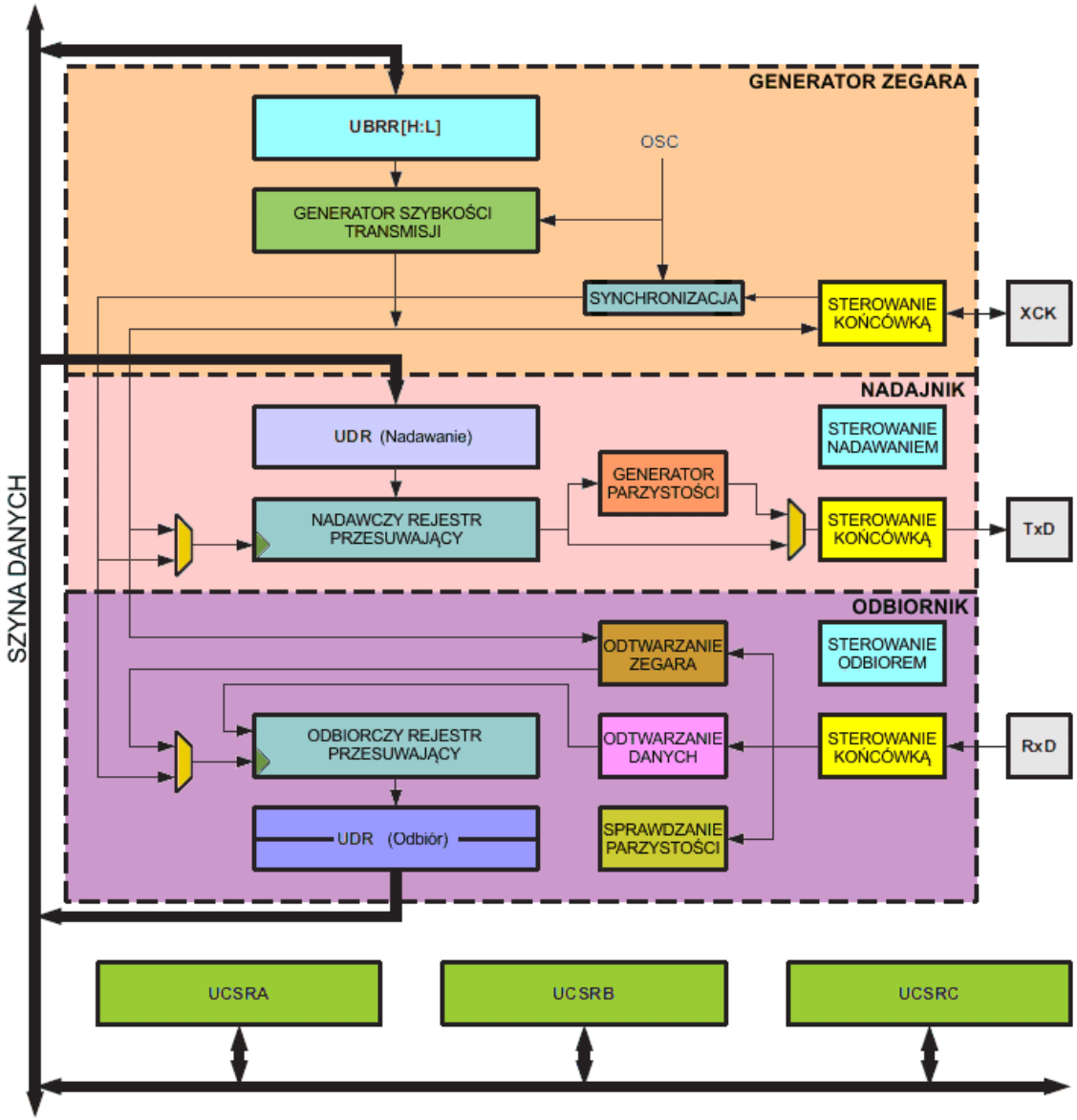
Interfejs USART - ramka transmisji

Co się dzieje gdy odbiornik próbkuje sygnał zbyt wolno:



Interfejs USART w ATmega32

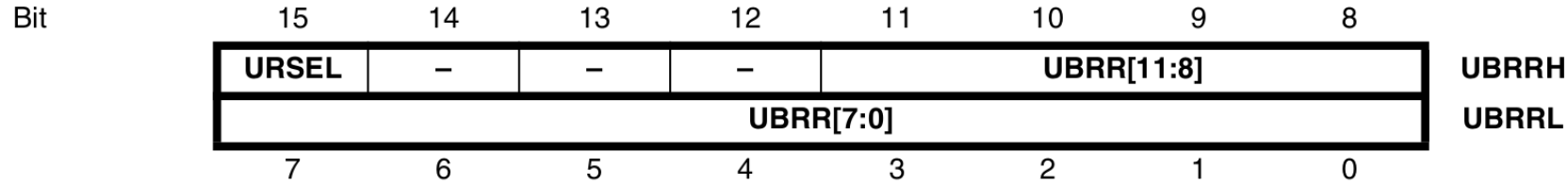
- Rejestr UBRR - wartość *baud rate*,
- Rejestr UDR - służy do przechowywania wysyłanej/odebranej danej (jeden znak),
- Rejestry UCSRA, UCSRB i UCSRC - służą do konfiguracji interfejsu,
- TxD oraz RxD to piny do komunikacji z innymi urządzeniami.



Źródło: `https://eduinf.waw.pl/inf/prg/009_kurs_avr/4346.php`

AVR USART - rejestr UBRR

- Rejestr 16-bitowy (UBRRL, UBRRH).
- W rejestrze UBRR nie podajemy liczby bodów, tylko wartość wyznaczoną według poniższego wzoru.



- Dygresja: Zapisując rejestr UBRRH, bit URSEL musi być $= 0$. Ponieważ dzieli on wspólny adres z rejestrem UCSRC (do którego można pisać gdy URSEL $= 1$).

Table 60. Equations for Calculating Baud Rate Register Setting

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

- Tryb U2X pozwala na zwiększenie prędkości dwukrotnie ale kosztem zmniejszenia ilości próbkowań poszczególnych bitów.

AVR USART - rejestr UBRR

Table 71. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000MHz				f_{osc} = 18.4320MHz				f_{osc} = 20.0000MHz			
	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	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max (1)	1Mbps		2Mbps		1.152Mbps		2.304Mbps		1.25Mbps		2.5Mbps	

1. UBRR = 0, Error = 0.0%

AVR USART - rejestr UDR (*data register*)

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
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	

- 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.
- Zatem aby wysłać znak wystarczy wpisać go do rejestru UDR.

AVR USART - rejestr UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **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*) - błąd parzystości.
- **U2X** - podwojenie szybkości transmisji (rzadziej próbkowany stan linii $\rightarrow$ mogą wystąpić błędy transmisji).
- **MPCM** (*MultiProcessor Communication Mode*) włącza tryb współpracy między mikrokontrolerami, gdzie np. 9 bit ramki $= 1$ oznacza adres, a 0 oznacza dane. Drugi układ, jeśli ma inny adres, nie będzie odbierał danych.

AVR USART - rejestr UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **RXCIE**: włączenie przerwania RXC (co oznacza, że odbiór się zakończył i znak znajduje się w rejestrze UDR),
- **TXCIE**: włączenie przerwania TXC (zakończono nadawanie znaku),
- **UDRIE**: włączenie przerwania UDRE (rejestr UDR jest wolny),
- **RXEN**: Włączenie odbiornika,
- **TXEN**: Włączenie nadajnika,
- **UCSZ2**: długość słowa,
- **RXB8, TXB8**: w trybie 9-bitowym tu znajdują się bity nr 8 (licząc od 0) przy odbiorze i przy transmisji.

Table 66. UCSZ Bits Settings

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

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,
- **UMP1, UMP0**: 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 ubrr_value )
{
    /*Prędkość transmisji*/
    UBRRH = (unsigned char)(ubrr_value>>8);
    UBRRL = (unsigned char)ubrr_value;

    /*Format ramki: słowo=8bitów, 2 bity stopu*/
    UCSRC =(1<<URSEL)|(1<<USBS)|(1<<UCSZ1)|(1<<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 ubrr_value )
{
    /* prędkość transmisji */
    UBRRH = (unsigned char)(ubrr_value>>8);
    UBRRL = (unsigned char)ubrr_value;

    /* Format ramki: słowo=8bitów, 2 bity stopu */
    UCSRC=(1<<URSEL)|(1<<USBS)|(1<<UCSZ1)|(1<<UCSZ0);

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

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

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

Wysyłanie i odbiór danej (przerwania)

- Przerwanie (USART_RXC_vect) będzie wywoływane tak długo, jak będą nieodebrane dane w rejestrze UDR.
- Procedura przerwania MUSI odczytać daną lub wyłączyć możliwość przerwań.

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

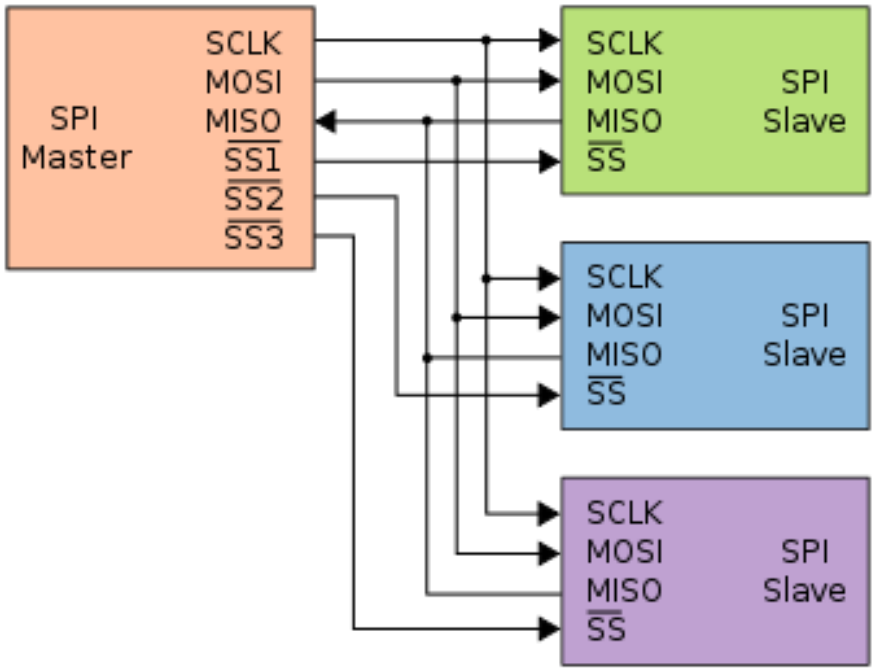
- Przerwanie (USART_UDRE_vect) będzie wywoływane tak długo, jak UDR będzie pusty.
- Procedura przerwania MUSI zapisać nową daną do UDR albo wyłączyć możliwość przerwań.

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

Interfejs SPI

- SPI - *Serial Peripherial Interface*.
- Opracowany przez firmę Motorola.
- Interfejs synchroniczny, czteroprzewodowy co oznacza, że w trakcie transmisji będzie przekazywany sygnał zegarowy, którego zbocza będą informowały, że odbiornik ma sprawdzać stan linii.
- SPI jest wykorzystywane do łączenia układów scalonych.
- Transmisja typu duplex czyli jednoczesna transmisja w obu kierunkach.
- Dwa rodzaje urządzeń na magistrali:
  - Układ *Master* - steruje przepływem danych na magistrali.
  - Układ *Slave* - układ wysyłający i odbierający dane.
- Gwarantowana prędkość transmisji: 2,1 Mbd (niekiedy nawet do 10 Mbd). Mbd - megabod.

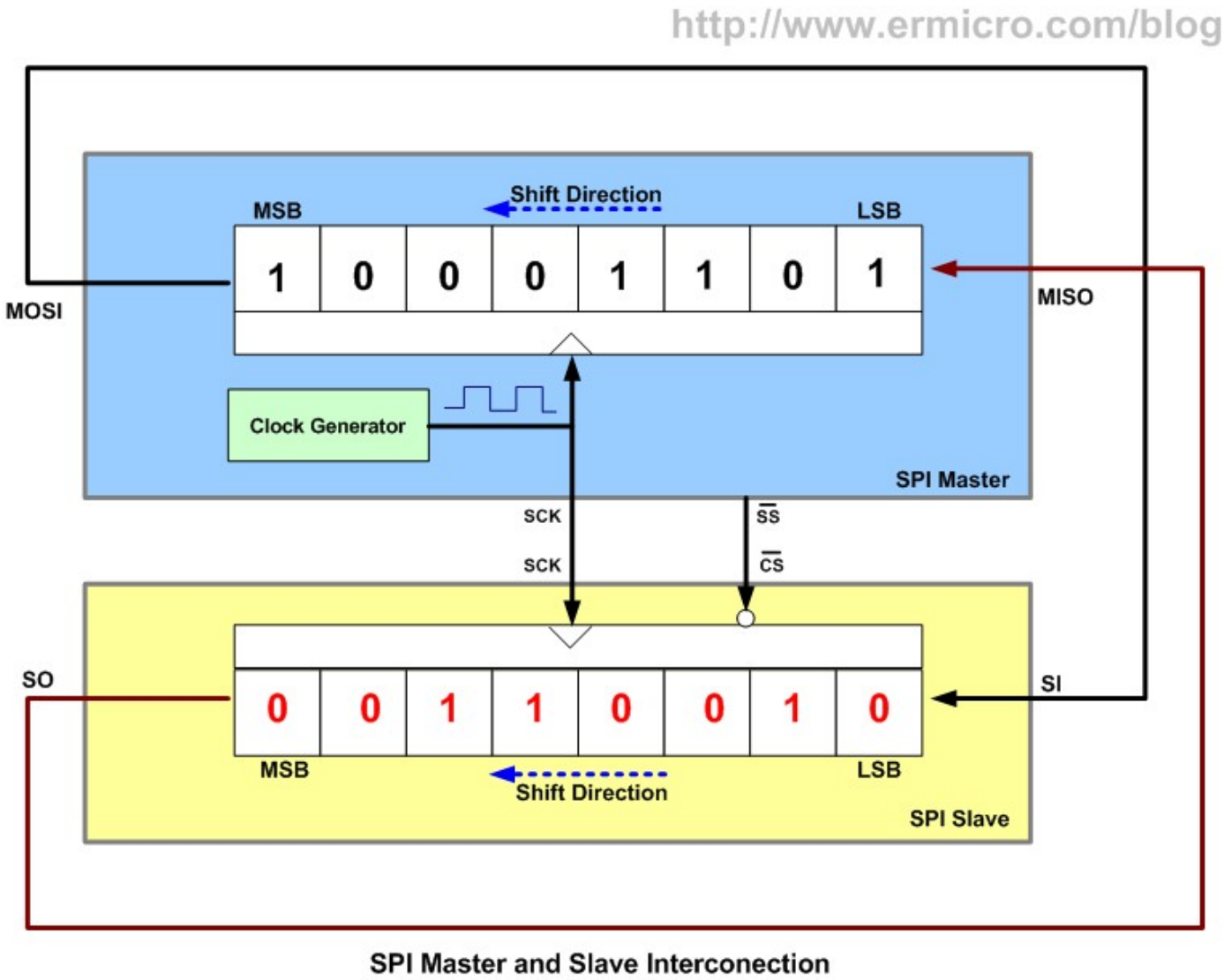
Podłączanie urządzeń do magistrali, linie interfejsu



Źródło: https://commons.wikimedia.org/wiki/File:SPI_three_slaves.svg

- 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. Aktywacja układu sygnałem niskim.

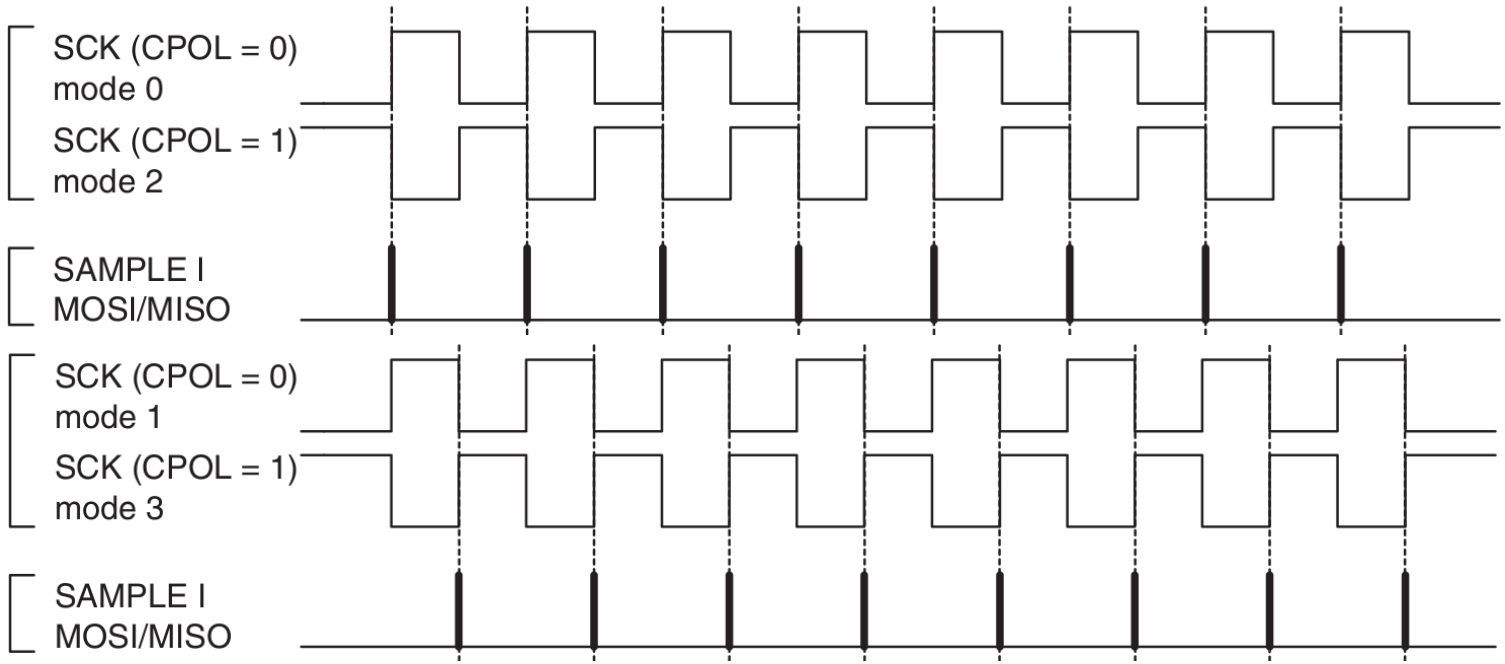
Dane są jednocześnie wysyłane i odbierane (full-duplex)



SPI Master and Slave Interconection

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 maja 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. Domyślnie wysyłany jest MSB pierwszy czyli odwrotnie niż dla RS232.
- 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) zboczu zegara.
- SPR1:0 : częstotliwość próbkowania.

SPI2X	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 i SPDR

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.

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);
}
```

- 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;
}
```

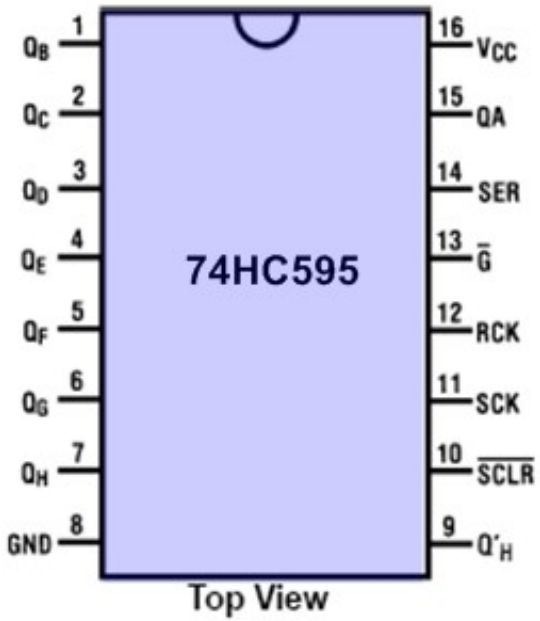
- Inicjalizacja modułu SPI w trybie Slave:

```
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 zatrzaskami 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. zatrzasku.

http://www.ermicro.com/blog



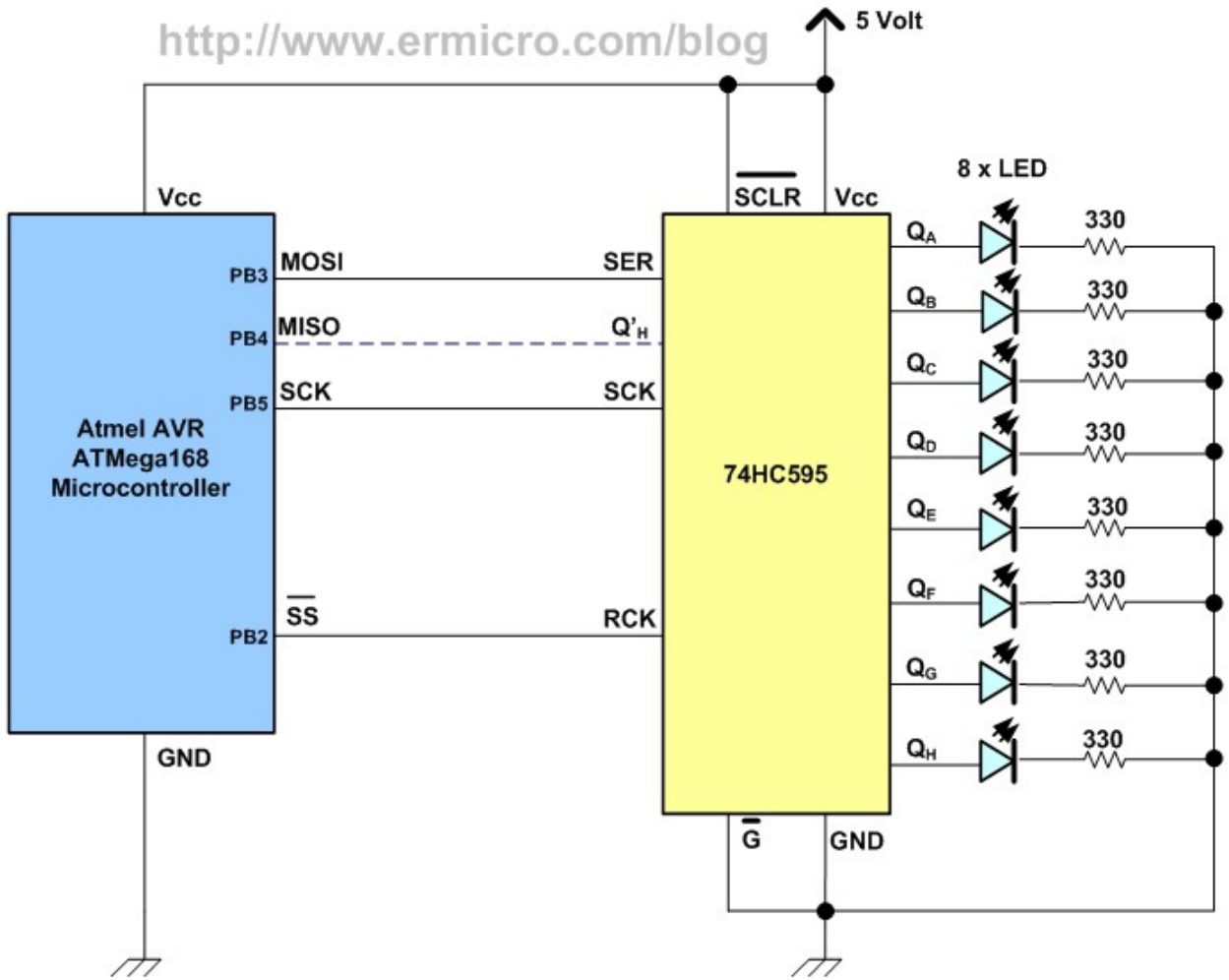
Top View

Truth Table

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

H – Logical High, L – Logical Low, X - Don't Care
↑ - Rising Clock Pulse

74HC595 8-Bit Shift Registers with Output Latches



Atmel AVR ATMega168 and 74HC595 8-Bit Shift Registers Circuit Diagram

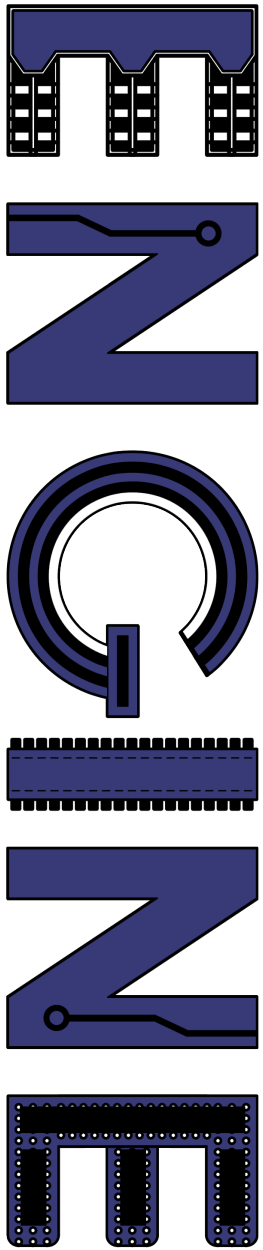
- Linia MOSI będzie przesyłała dane do rejestru przesuwnego 74HC595.
- Zatrzaskiwanie danych linią SS.

```c
#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 zakonczenie trans.
    PORTB |= (1<<SS); // zbocze narastające zatrzaskuje wartości
    //rejestru do wyjść Qa-Qh
    _delay_us(1); // czekamy
    PORTB &= ~(1<<SS); // przywracamy stan niski na linii zatrzasku
}

int main(void) {
    InitSPI(); // inicjalizacja SPI - zrobiliśmy to wcześniej
    while(1) {
        cnt=1; //counter
        while(cnt) {
            SendSPI( cnt );
            _delay_ms(100);
            cnt << = 1; //przesuwanie 1 by zapalić jedną diodę, która
            //będzie się przesuwać
        }
    }
}
```

Dziękuję za uwagę.



Pierwotna wersja wykładu jest dziełem:
dr hab. Piotra Fronczaka, mgr inż. Marcina Zaręby