



**Dariusz Makowski**

**Katedra Mikroelektroniki i Technik  
Informatycznych**

**tel. 631 2720**

**[dmakow@dmcs.pl](mailto:dmakow@dmcs.pl)**

**<http://neo.dmcs.p.lodz.pl/sw>**



- Informacje o zwolnieniu z egzaminu
- Hasło do pliku z informacją o zwolnieniu: sW2019



## Zakres przedmiotu

- ◆ Systemy mikroprocesorowe, systemy wbudowane
- ◆ Laboratorium
- ◆ Rodzina procesorów ARM
- ◆ Urządzenia peryferyjne
- ◆ **Interfejsy w systemach wbudowanych**
- ◆ Pamięci i dekodery adresowe
- ◆ Programy wbudowane na przykładzie procesorów ARM
- ◆ Metodyki projektowania systemów wbudowanych
- ◆ Systemy czasu rzeczywistego



# Interfejsy w systemach wbudowanych



## Definicje podstawowe (3)

### ★ Pamięć komputerowa (ang. Computer Memory)

Pamięć komputerowa to urządzenie elektroniczne lub mechaniczne służące do przechowywania danych i programów (systemu operacyjnego oraz aplikacji).

### ★ Urządzenia zewnętrzne, peryferyjne (ang. Peripheral Device)

Urządzenia elektroniczne dołączone do procesora przez magistrale systemową lub interfejs. Urządzenia zewnętrzne wykorzystywane są do realizowania specjalizowanej funkcjonalności systemu.

### ★ Magistrala (ang. bus)

Połączenie elektryczne umożliwiające przesyłanie danym pomiędzy procesorem, pamięcią i urządzeniami peryferyjnymi. Magistra systemowa zbudowane jest zwykle z kilkudziesięciu połączeń elektrycznych (ang. Parallel Bus) lub szeregowego połączenia (ang. Serial Bus).

### ★ Interface (ang. Interface)

Urządzenie elektroniczne lub optyczne pozwalające na komunikację między dwoma innymi urządzeniami, których bezpośrednio nie da się ze sobą połączyć.



## Współpraca procesora z urządzeniami peryferyjnymi

### Interfejsy dostępne w procesorach rodziny ARM:

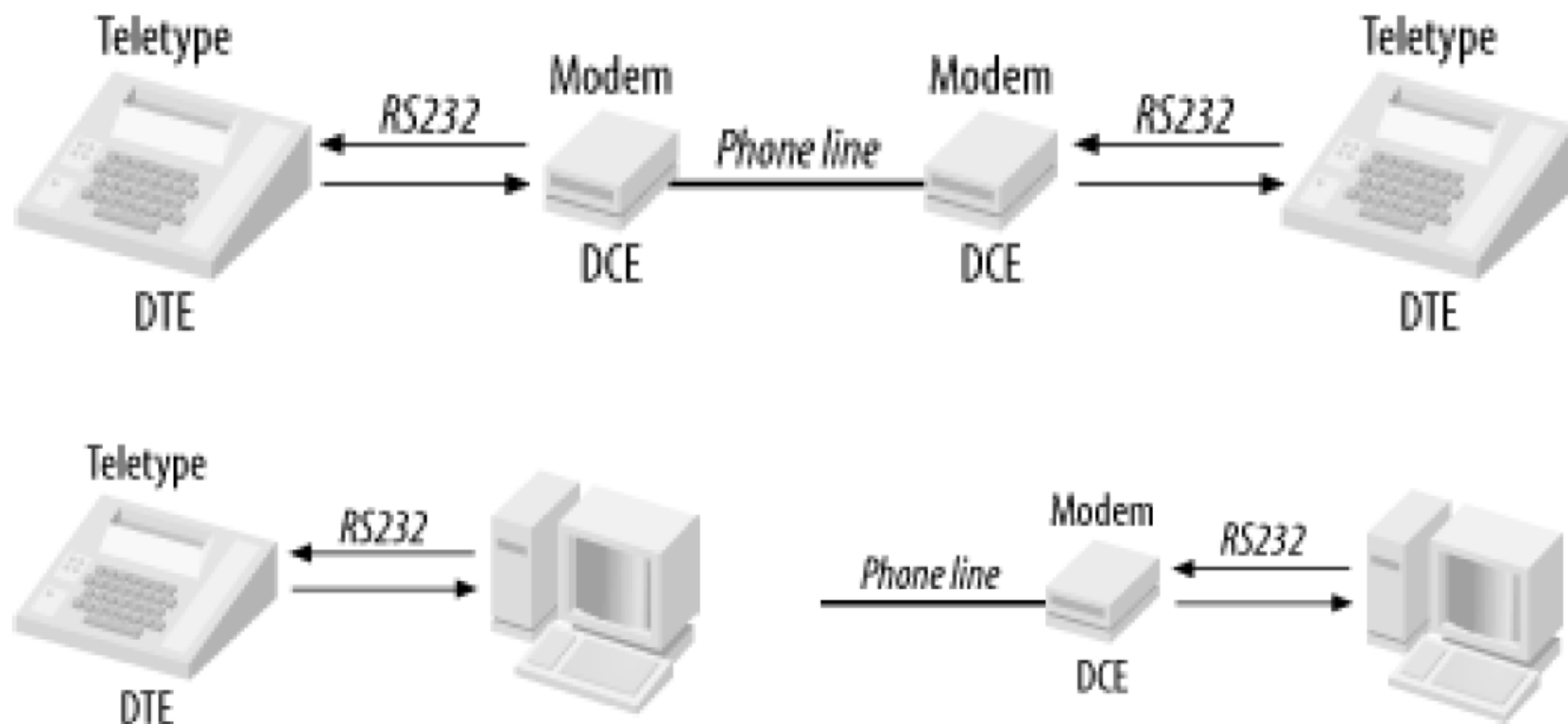
- Interfejs równoległy PIO (zwykle 32 bity),
- Interfejsy szeregowe:
  - Interfejs DBGU - zgodny ze standardem EIA RS232,
  - Interfejs uniwersalny USART,
  - Interfejs Serial Peripheral Interface (SPI),
  - Interfejs Synchronous Serial Controller (SSC)
  - Interfejs I2C, Interfejs Two-wire Interface (TWI),
  - Interfejs Controlled Area Network (CAN),
  - Interfejs Universal Serial Bus (USB),
  - Interfejs Ethernet 10/100.



# Moduł transceivera szeregowego UART (Universal Asynchronous Receiver/Transmitter module)



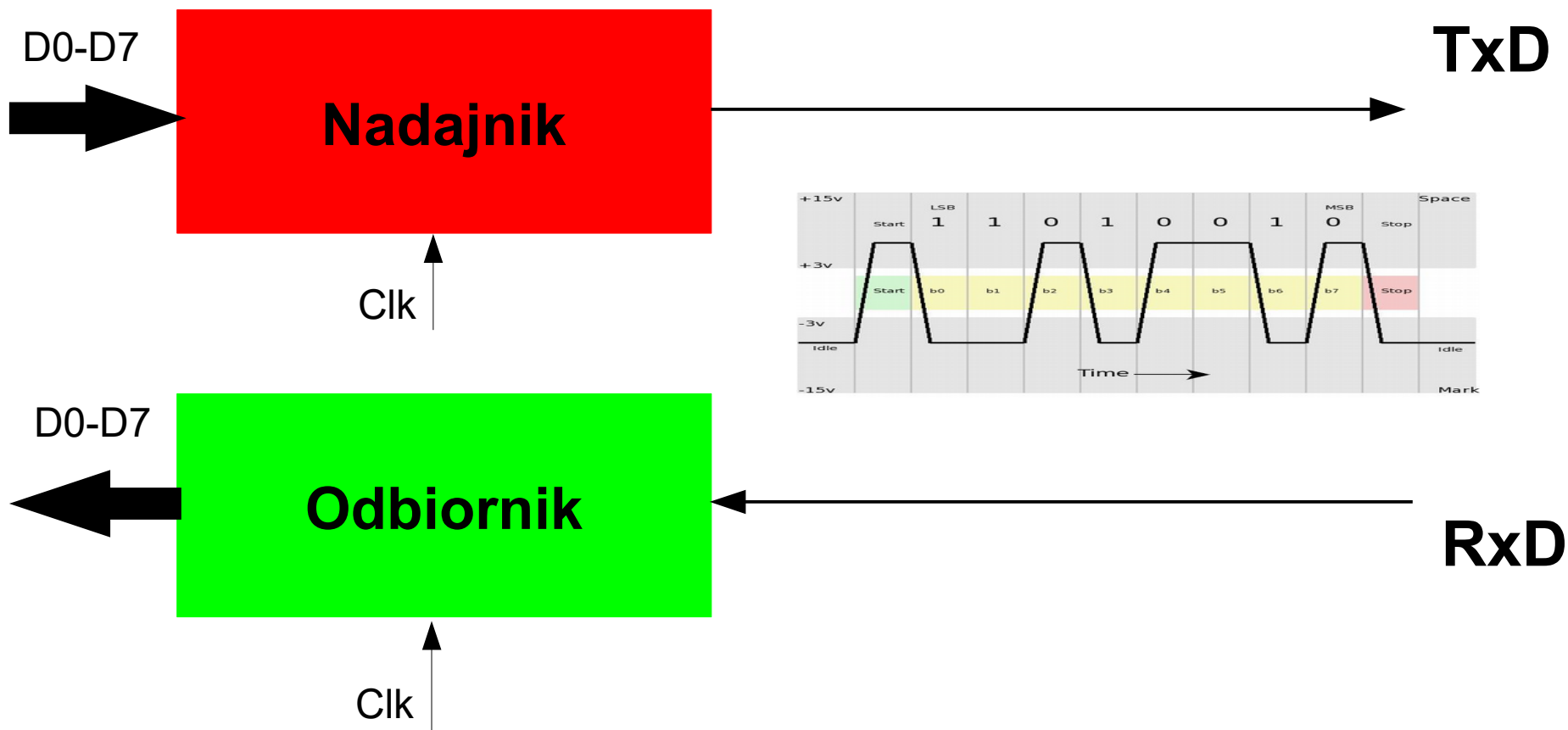
# Interfejs szeregowy EIA RS232







## Rejestr przesuwny

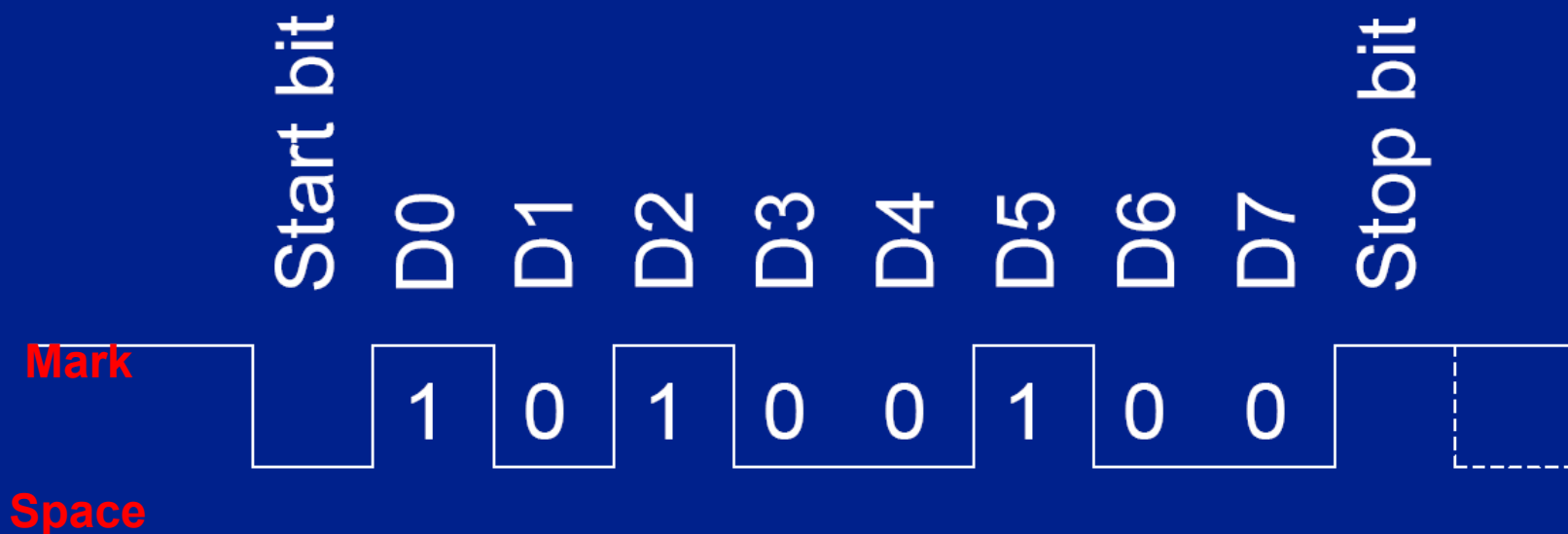




## Ramka danych transmitera UART (1)

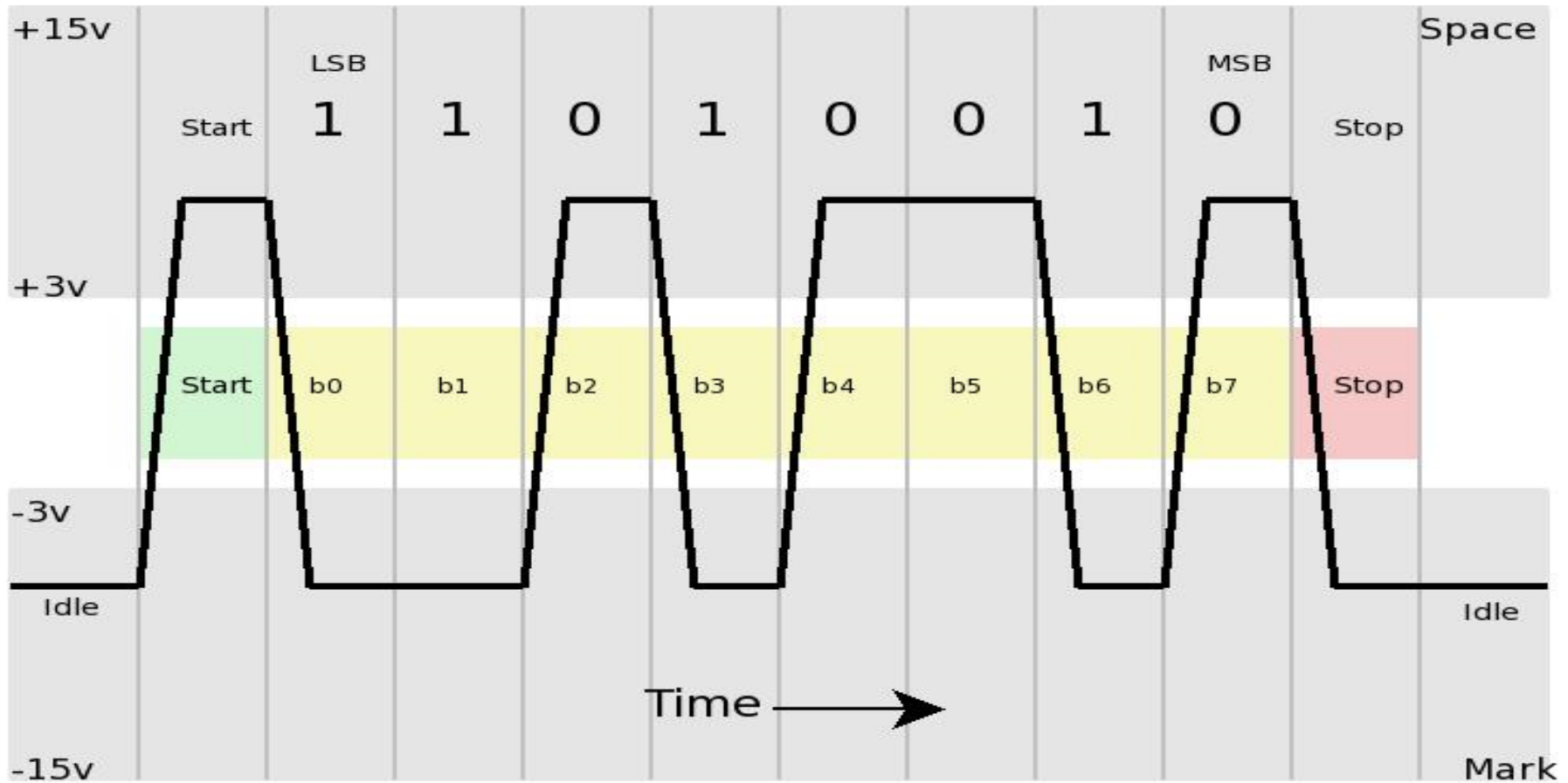
## Asynchronous 8 bit waveform example

- Data is H'25' = B'00100101'





## Ramka danych transmitera UART (2)

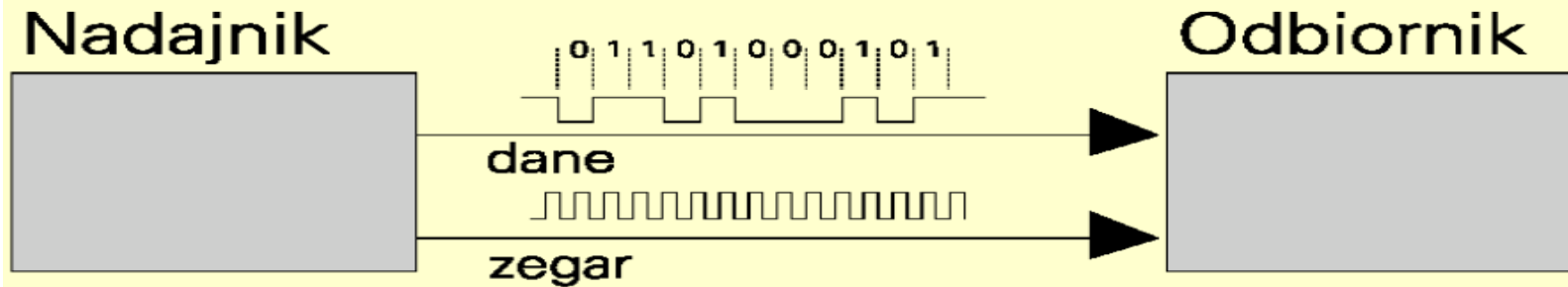


Przesyłana dana: 0100.1011b = 0x4B

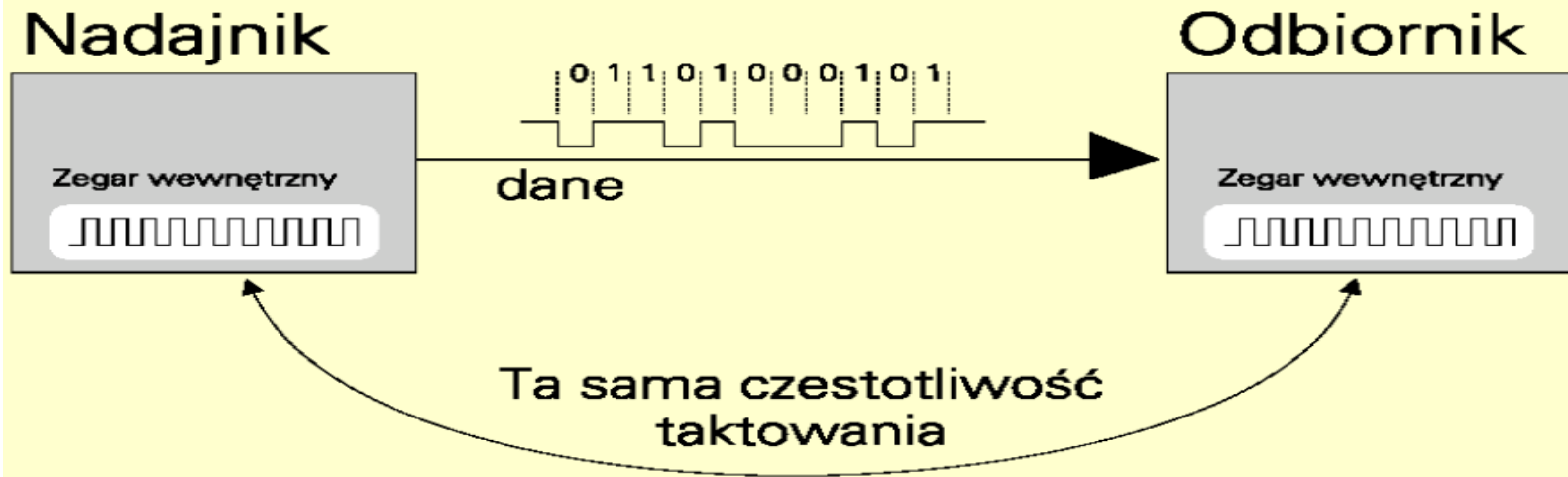


# Transmisja synchroniczna vs asynchroniczna?

## a) transmisja synchroniczna

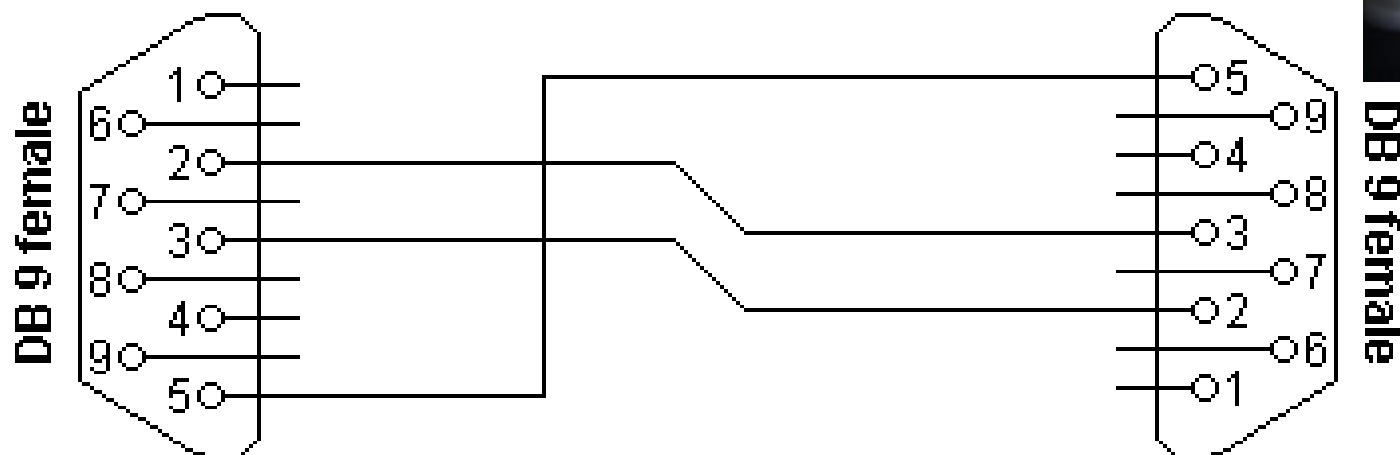


## b) transmisja asynchroniczna





# Kabel null-modem EIA 232

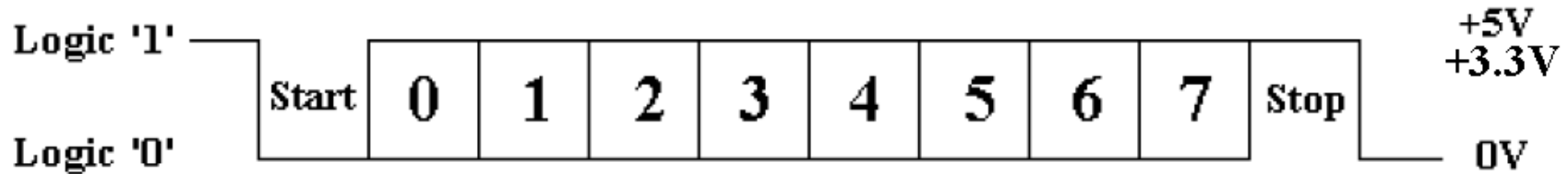


Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	Signal ground

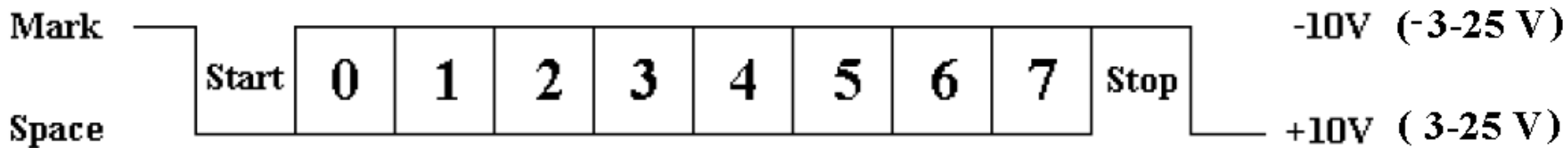


# Poziomy napięć określone przez standard EIA 232

## Wyjście procesora



## Standard EIA RS 232

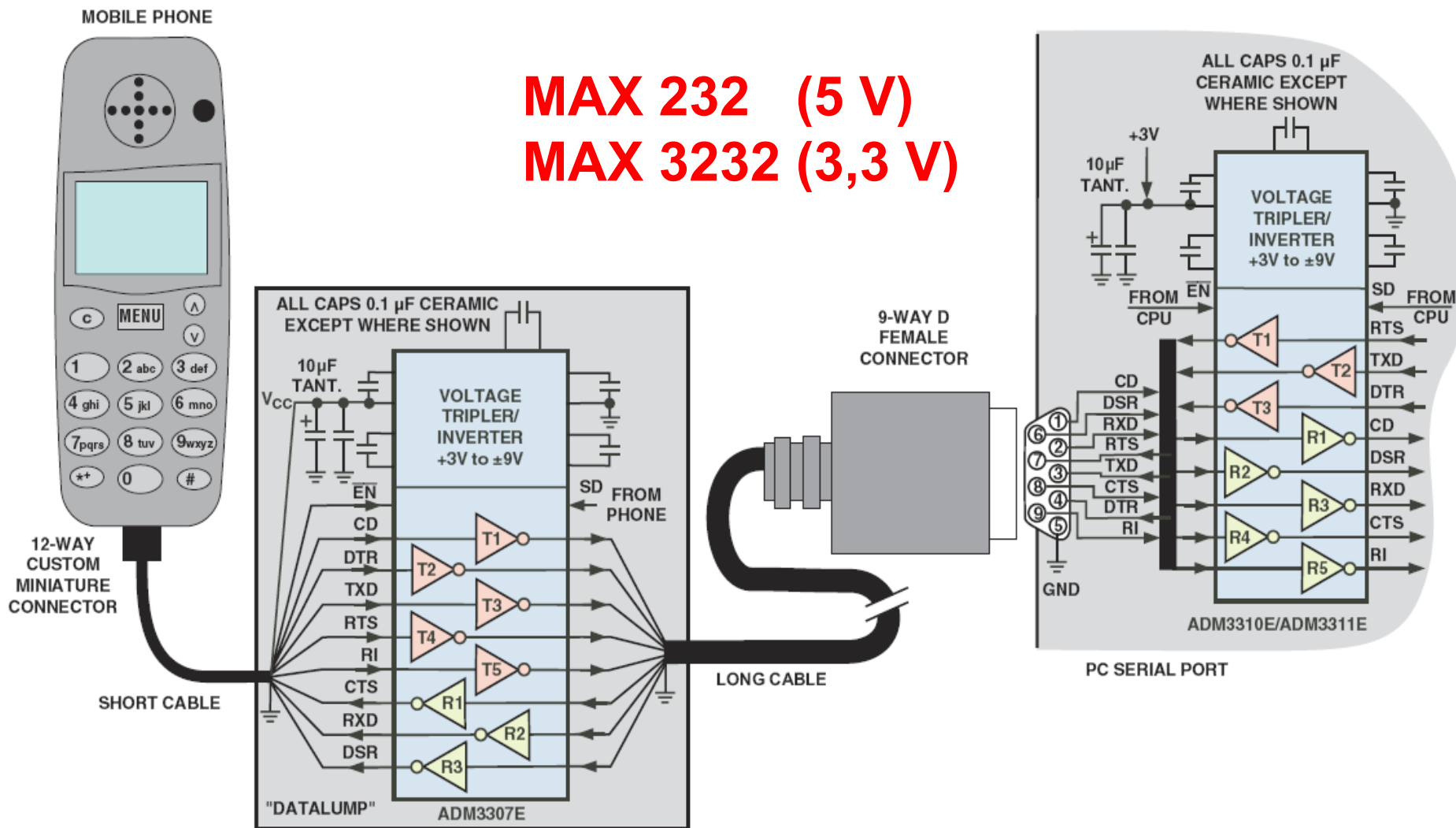


RS-232 Logic Waveform



# Konwerter poziomów napięć

**MAX 232 (5 V)**  
**MAX 3232 (3,3 V)**





# Programy do komunikacji z wykorzystaniem standardu EIA RS232

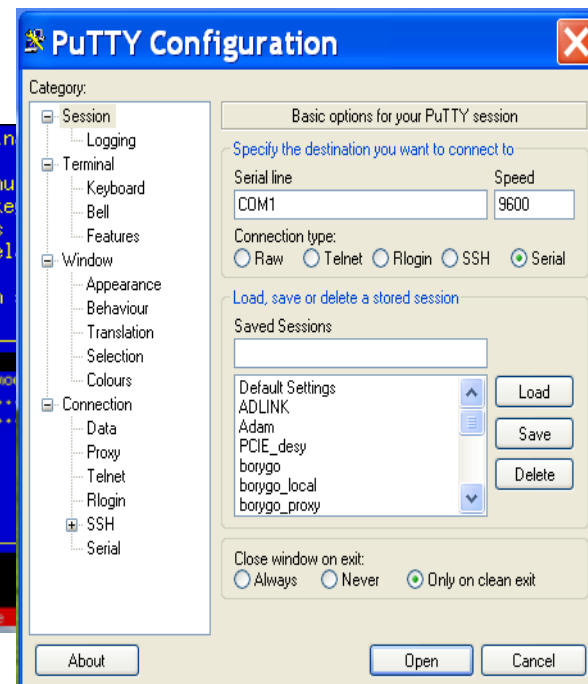
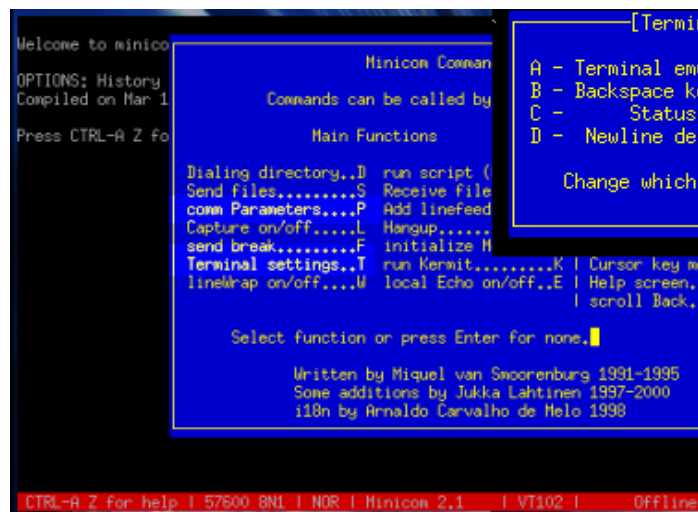
Program Hyper terminal

Program minicom

Program ssh

Program Terminal

(<http://www.elester-pkp.com.pl/index.php?id=92&lang=pl&zoom=0>)







# AT91SAM9263 – moduł diagnostyczny DBGU (rozdział 30)



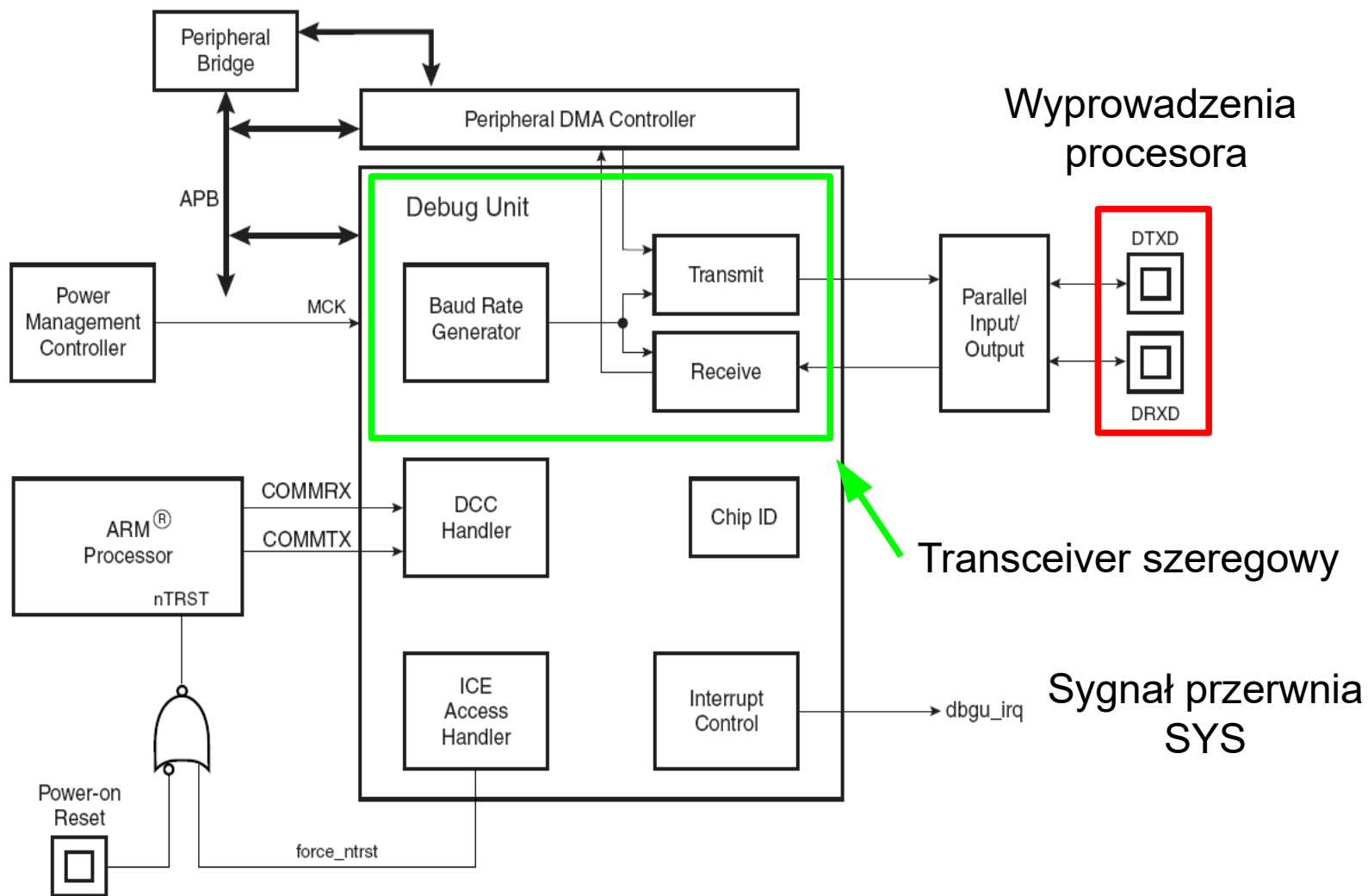
## Port szeregowy jako interfejs diagnostyczny

Cechy portu diagnostycznego DBGU (DeBuG Unit):

- ▶ Asynchroniczna transmisja danych zgodna ze standardem RS232 (8 bitów danych, jeden bit parzystości z możliwością wyłączenia),
- ▶ Możliwość zgłaszania przerw systemowych współdzielonych (PIT, RTT, WDT, DMA, PMC, RSTC, MC),
- ▶ Analiza poprawności odebranych ramek,
- ▶ Sygnalizacja przepełnionego bufora TxD lub RxD,
- ▶ Trzy tryby diagnostyczne: zdalny loopback, lokalny loopback oraz echo,
- ▶ Maksymalna szybkość transmisji rzędu 1 Mbit/s,
- ▶ Możliwość komunikacji z rdzeniem procesora COMMRx/COMMTx.

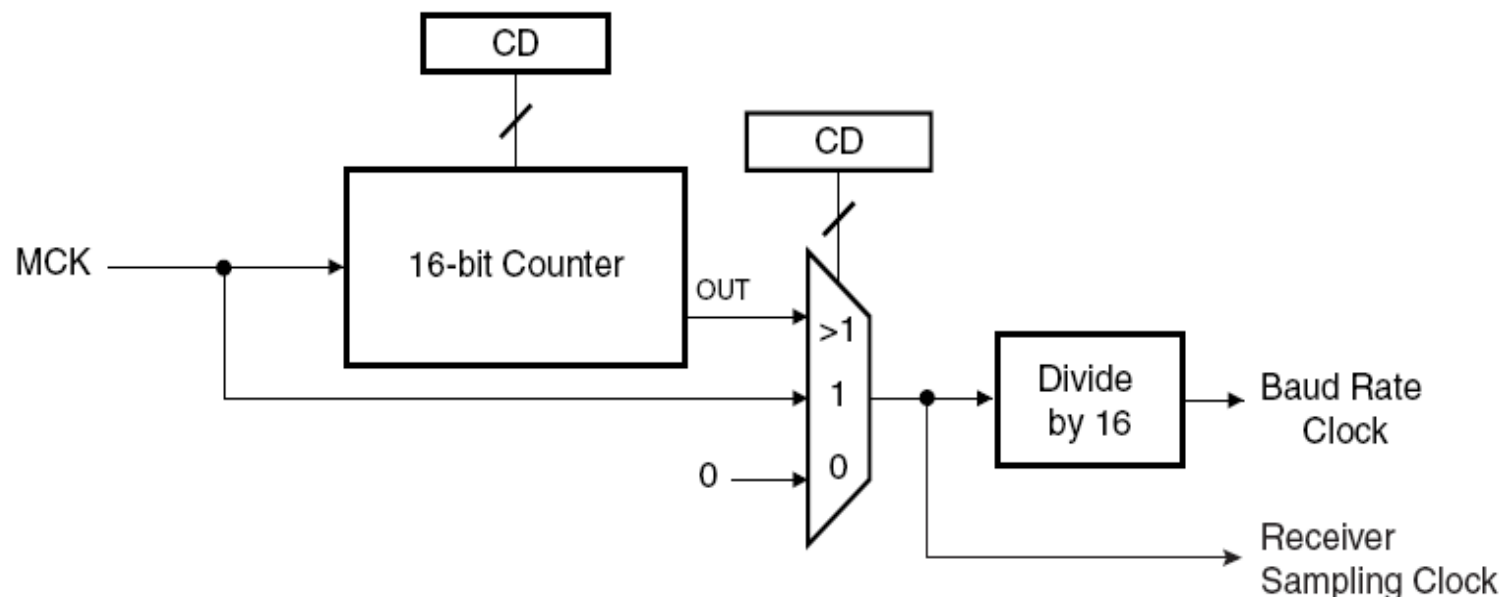


# Schemat blokowy portu DBGU procesora ARM 9





# Szybkość transmisji



Generator sygnału zegarowego odpowiedzialnego za szybkość transmisji (ang. Baud Rate).

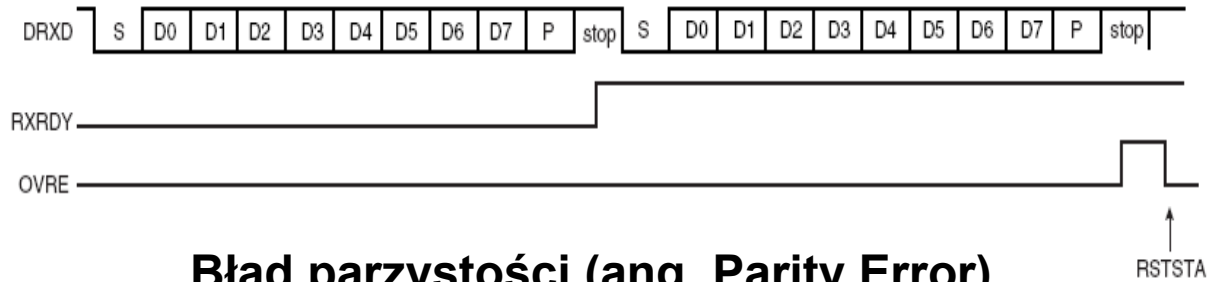
Szybkość transmisji danych wyrażona jest wzorem:

**Baud Rate =  $MCK / (16 \times CD)$** , gdzie CD (Clock Divisor) jest polem rejestru DBGU\_BRGR

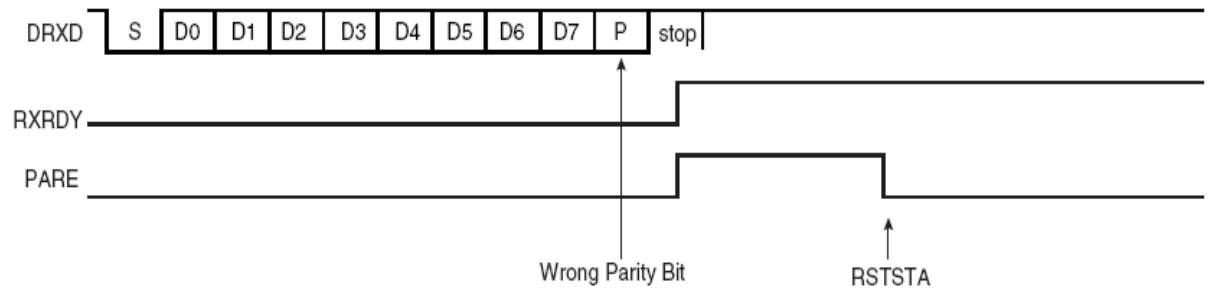


# Błędy podczas transmisji danych

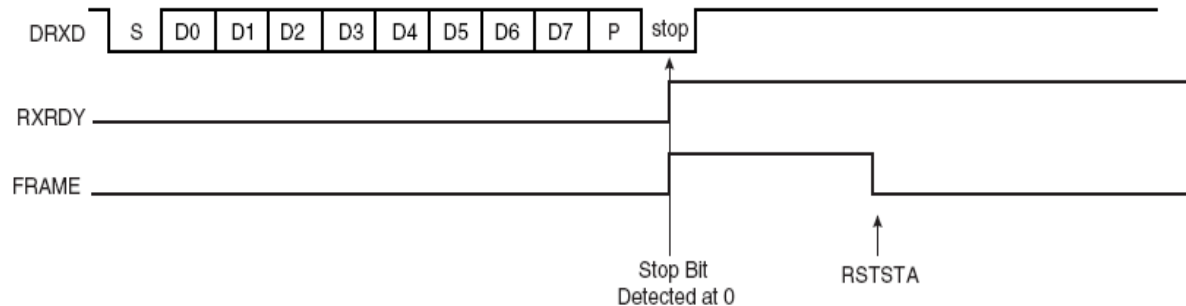
## Przepełnienie bufora odbiorczego BGU\_RHR (ang. Buffer Overflow)



## Błąd parzystości (ang. Parity Error)



## Błąd ramki (ang. Frame Error)





## Konfiguracja portu DBGU

```
static void Open_DBGU (void){
```

1. Wyłącz przerwania od portu DBGU, rejestr AT91C\_BASE\_DBGU->DBGU\_IDR
2. Resetuj i wyłącz odbiornik AT91C\_BASE\_DBGU->DBGU\_CR
3. Resetuj i wyłącz nadajnik AT91C\_BASE\_DBGU->DBGU\_CR
4. Konfiguracja portów wejścia-wyjścia jako porty RxD i TxD DBGU, rejestry AT91C\_BASE\_PIOC->PIO\_ASR oraz AT91C\_BASE\_PIOC->PIO\_PDR
5. Konfiguracja szybkości transmisji portu szeregowego AT91C\_BASE\_DBGU->DBGU\_BRGR
6. Konfiguracja trybu pracy, tryb normalny bez przystości (8N1), rejestr AT91C\_BASE\_DBGU->DBGU\_MR, flagi AT91C\_US\_CHMODE\_NORMAL, AT91C\_US\_PAR\_NONE;
7. Skonfiguruj przerwania jeżeli są wykorzystywane: Open\_DBGU\_INT()
8. Włącz odbiornik, rejestr AT91C\_BASE\_DBGU->DBGU\_CR
9. Włącz nadajnik, rejestr AT91C\_BASE\_DBGU->DBGU\_CR

```
}
```



## Odczyt i zapis danych do portu DBGU

```
void dbg_u_print_ascii (const char *Buffer)
{
    while ( data_are_in_buffer ) {
        while ( ...TXRDY... );           /* wait until Tx buffer busy – check TXRDY flag */
        DBGU_THR = ...                    /* write a single char to Transmitter Holding Register */
    }
}
```

```
void dbg_u_read_ascii (const char *Buffer, unsigned int Size){
    do {
        While ( ...RXRDY... );           /* wait until data available */
        Buffer[...] = DBGU_RHR;           /* read data from Receiver Holding Register */
    } while ( ...read_enough_data... )
}
```



# Buforowanie danych





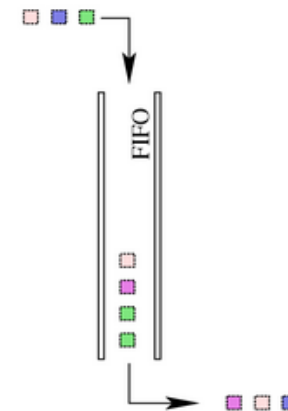
## Struktura stosu (1)

**Stos (ang. stack lub LIFO Last-In, First-Out) -** liniowa struktura danych, w której dane odkładane są na wierzch stosu i z wierzchołka stosu są zdejmowane. Ideę stosu danych można zilustrować jako stos położonych jedna na drugiej książek – nowy egzemplarz kładzie się na wierzch stosu i z wierzchu stosu zdejmuje się kolejne egzemplarze. Elementy stosu poniżej wierzchołka stosu można wyłącznie obejrzeć, aby je ściągnąć, trzeba najpierw po kolei ściągnąć to, co jest nad nimi

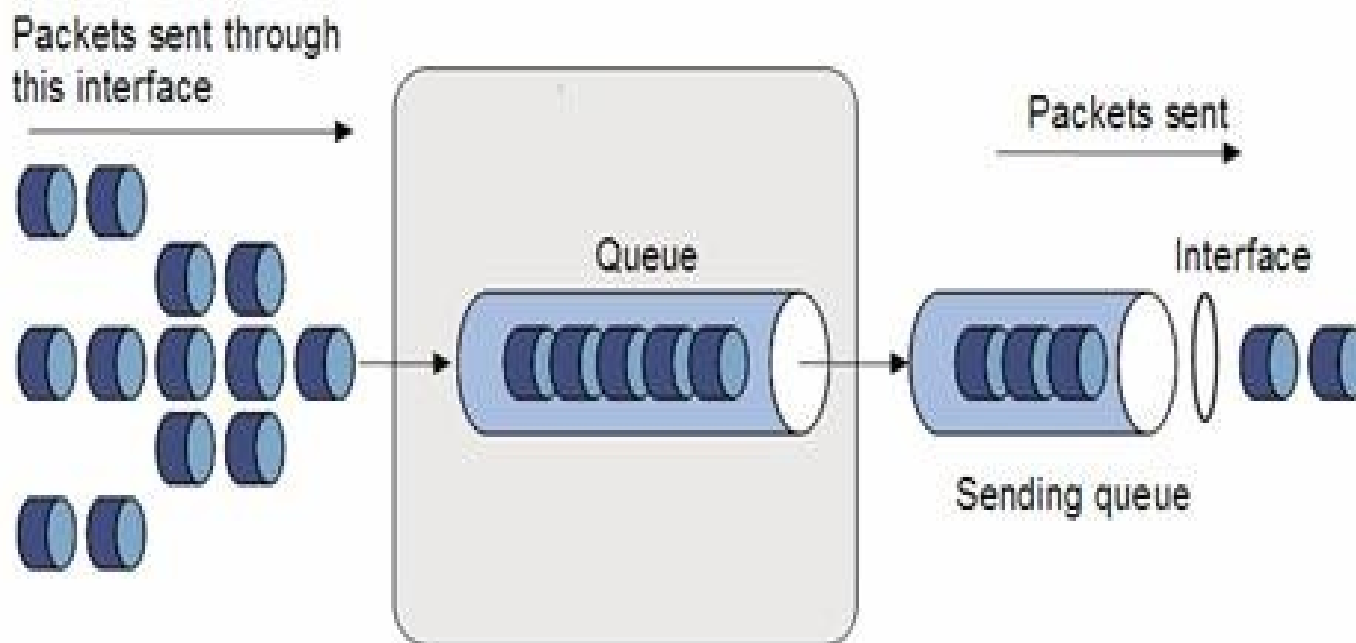
**FIFO (ang. First In, First Out) -** przeciwieństwem stosu LIFO jest kolejka, bufor typu FIFO (pierwszy na wejściu, pierwszy na wyjściu), w którym dane obsługiwane są w takiej kolejności, w jakiej zostały dostarczone (jak w kolejce do kasy)



First-in First-out (FIFO)



## Kolejka FIFO (1)



- ★ Dane do kolejki FIFO mogą być wpisywane przez kilka niezależnych aplikacji, wątków lub urządzeń. W takiej sytuacji dostęp do kolejki kontrolowany jest przez Semafor (zmienna globalna).
- ★ Dane zgromadzone w kolejce wysyłane są w kolejności w jakiej zostały wpisane.

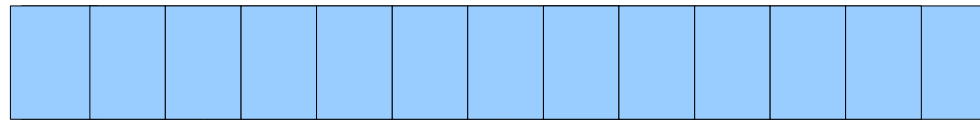


## Kolejka FIFO (2)

### Dane w kolejce FIFO

Adres w pamięci: 0xffD50

0xffD50 + size - 1



Tail

Head

### Zapisz danej do kolejki FIFO:

- ★ Zwiększ wskaźnik Head o jeden, zapisz daną.

### Odczyt danej z kolejki FIFO:

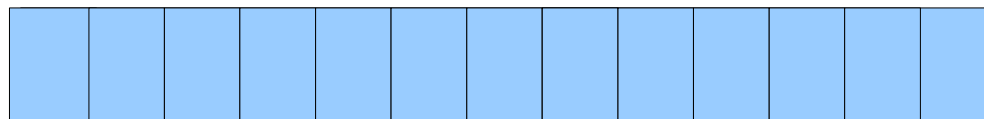
- ★ Odczytaj daną, zwiększ wskaźnik Tail o 1.

W przypadku, gdy Tail lub Head wskazuje na ostatni dostępny element kolejki zamiast inkrementacji wskaźnik jest zerowany. Pozwala to na płynne przesuwanie wskaźników – bufor kołowy (ang. circular buffer).



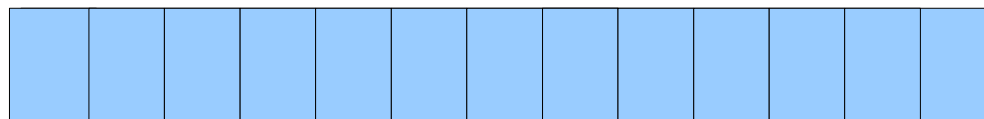
# Kolejka FIFO (3)

Kolejka pusta  $T = H$



T H

Dane w kolejce, ilość danych =  $H - T$

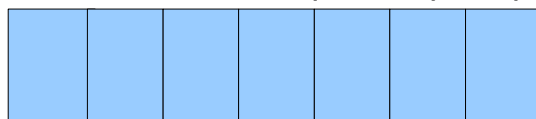


T

H

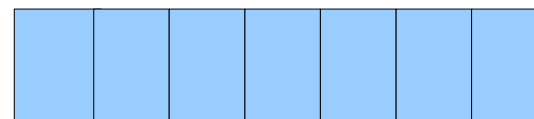
Brak miejsca w kolejce

$(T = 0) \& (H = \text{Size})$  lub  $T - H = 1$



T

H



H

T



## Kolejka FIFO – implementacja w C (1)

```
#define BUFFERSIZE  0xFF                /* FIFO buffer size and mask */
typedef struct FIFO {
    char buffer [BUFFERSIZE+1];
    int head;
    int tail;
};
void FIFO_Init (struct FIFO *Fifo);
void FIFO_Empty (struct FIFO *Fifo);
int FIFO_Put (struct FIFO *Fifo, char Data);
int FIFO_Put (struct FIFO *Fifo, char *Data);

void FIFO_Init (struct FIFO *Fifo){
    Fifo->head=0;
    Fifo->tail=0;

    /* optional: initialize data in buffer with 0 */
}
}
```



## Kolejka FIFO – implementacja w C (2)

```
void FIFO_Empty (struct FIFO *Fifo){
    Fifo->head = Fifo->tail;                                /* now FIFO is empty*/
}

int FIFO_Put (struct FIFO *Fifo, char Data){
    if ((Fifo->tail-Fifo->head)==1 || (Fifo->tail-Fifo->head)==BUFFERSIZE){
        return -1; };                                       /* FIFO overflow */
    Fifo->buffer[Fifo->head] = Data;
    Fifo->head = Fifo->head++ & BUFFERSIZE;
    return 1;                                              /* Put 1 byte successfully */
}

int FIFO_Get (struct FIFO *Fifo, char *Data){
    If ((Fifo->head!=Fifo->tail)){
        *Data = Fifo->buffer[Fifo->tail];
        Fifo->tail = Fifo->tail++ & BUFFERSIZE;
        return 1;                                          /* Get 1 byte successfully */
    } else return -1;                                       /* No data in FIFO */
}
```



# Serial Peripheral Interface



# Serial Peripheral Interface

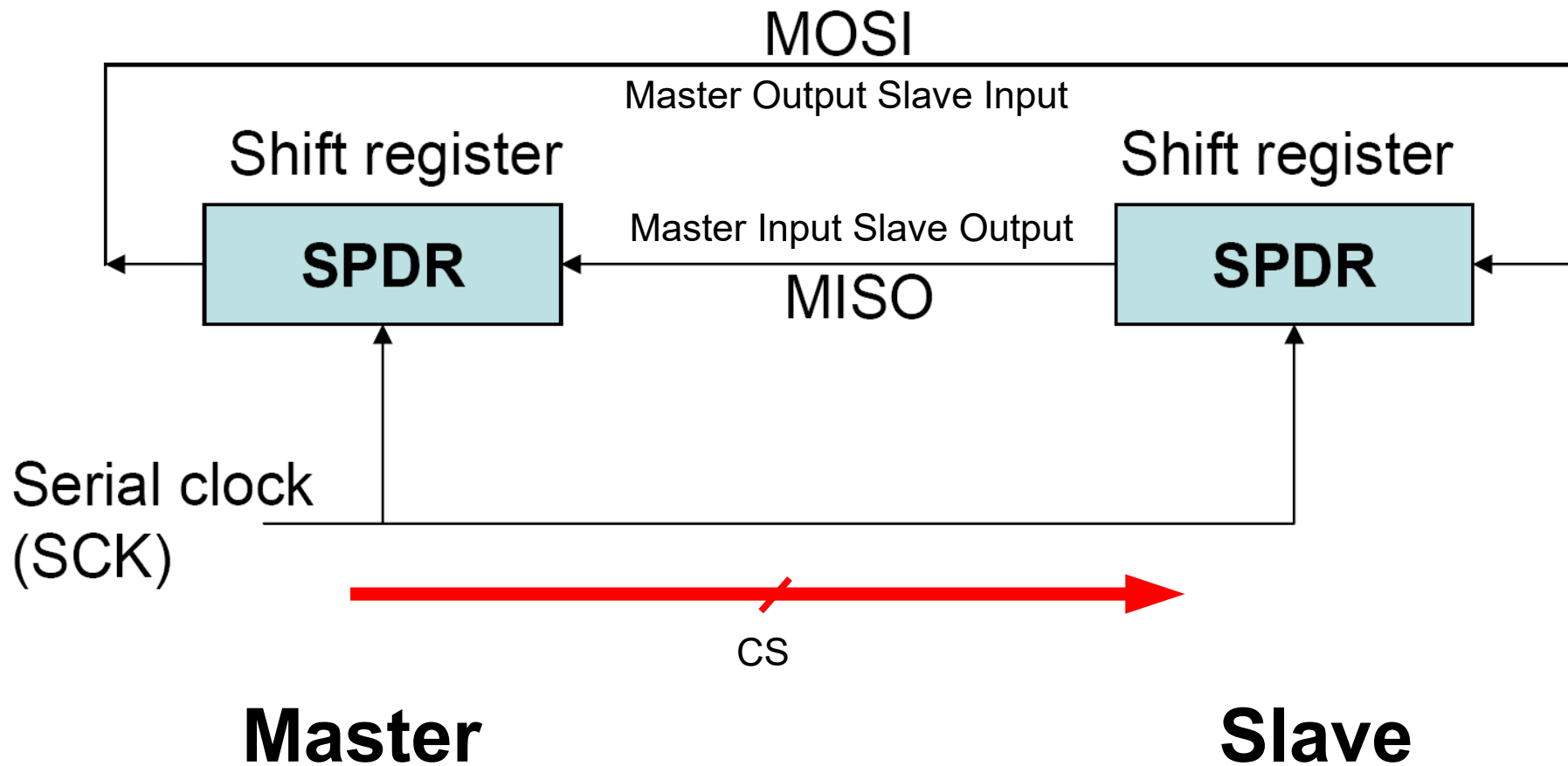
## Cechy interfejsu SPI:

- Szeregową transmisję synchroniczną,
- Transfer full duplex, master-slave lub master-multi-slave,
- Duża szybkość transmisji (>12 Mbit/s),
- Zastosowanie:
  - układy peryferyjne (ADC, DAC, RTC, EEPROM, termometry, itp),
  - sterowanie pomocnicze (matryca CCD z szybkim interfejsem równoległym),
  - karty pamięci z interfejsem szeregowym SD/SDHC/MMC.





# Serial Peripheral Interface





## Konfiguracja sygnału zegarowego:

### Polaryzacja zegara:

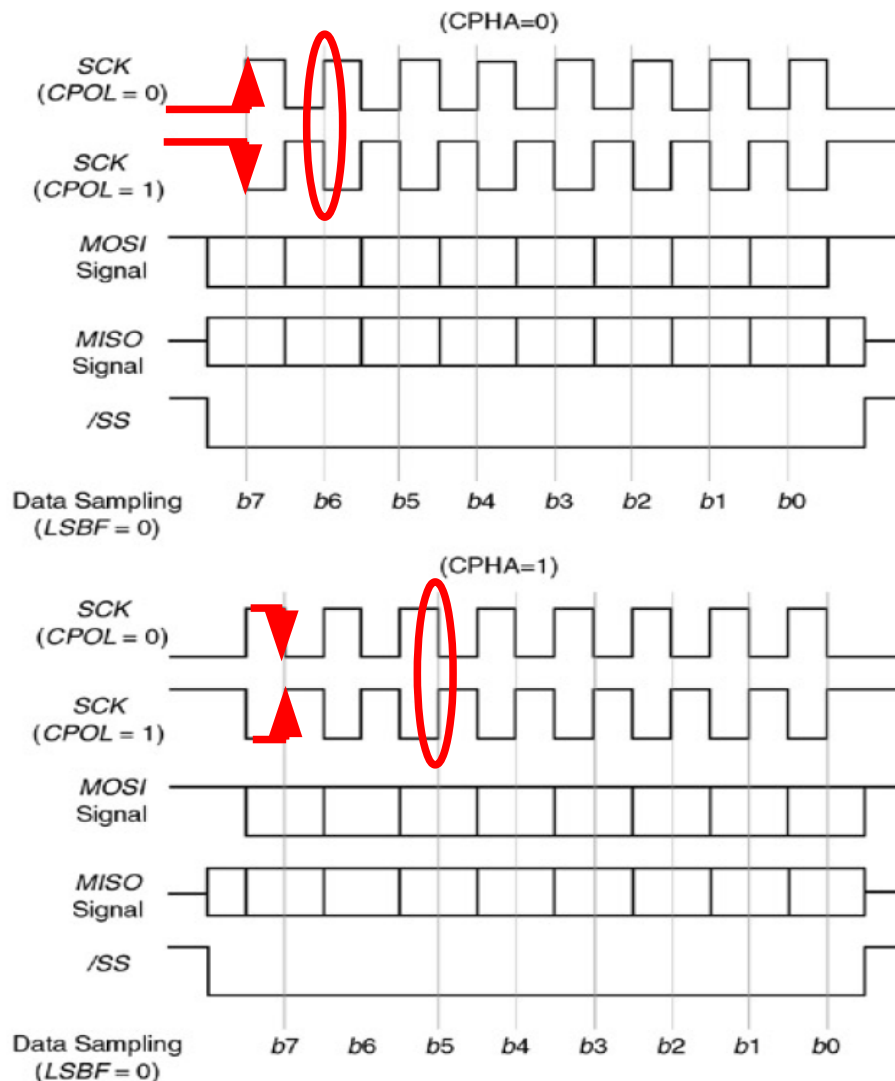
Polaryzacja **ujemna** **CPOL = 0**  
(**stan niski**, 8 impulsów zegara),

Polaryzacja **dodatnia** **CPOL = 1**  
(**stan wysoki**, 8 ujemnych impulsów zegara).

### Faza zegara:

Zerowa faza zegara (próbkiwanie na pierwszym zboczcu zegara),

Opóźniona faza zegara (próbkiwanie na drugim zboczcu zegara).

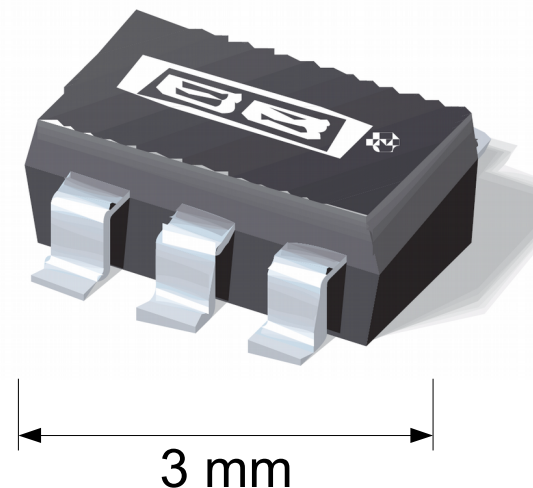




# Termometr z interfasem SPI

## TMP 121:

- Obudowa SOT 23-6,
- fclk mak. = 15 MHz
- Interfejs: SPI-Compatible Interface
- Rozdzielczość: 12-Bit + Sign, 0,0625°C
- Dokładność:  $\pm 1.5^{\circ}\text{C}$  od  $-25^{\circ}\text{C}$  do  $+85^{\circ}\text{C}$
- Pobór prądu w stanie uśpienia: 50 $\mu\text{A}$  (mak.)
- Zasilanie: 2,7V to 5,5V



D15	D14	D13	D12	D11	D10	D9	D8
T12	T11	T10	T9	T8	T7	T6	T5

D7	D6	D5	D4	D3	D2	D1	D0
T4	T3	T2	T1	T0	0	Z	Z

Table 1. Temperature Register

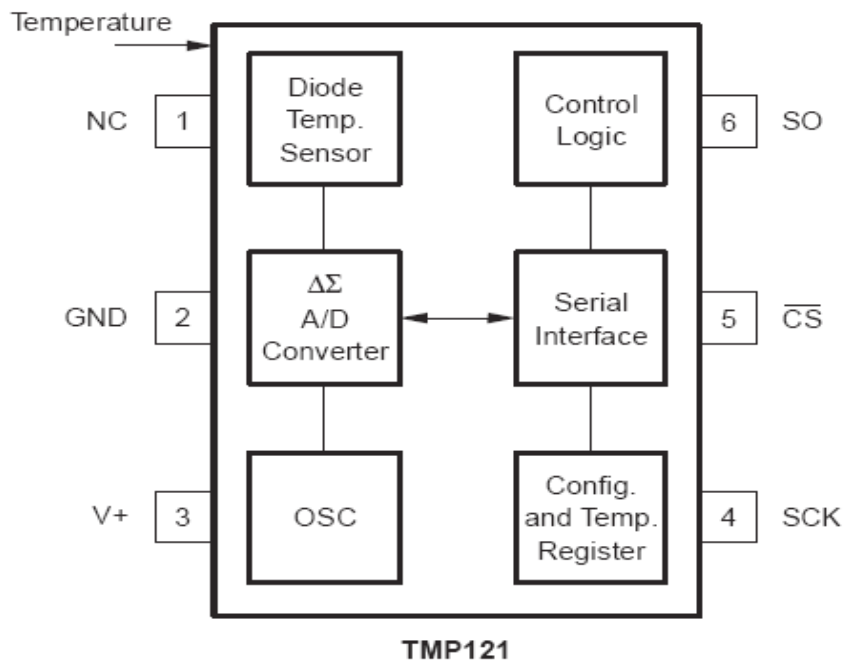
TEMPERATURE (°C)	DIGITAL OUTPUT <sup>(1)</sup> (BINARY)	HEX
150	0100 1011 0000 0000	4B00
125	0011 1110 1000 0000	3E80
25	0000 1100 1000 0000	0C80
0.0625	0000 0000 0000 1000	0008
0	0000 0000 0000 0000	0000
-0.0625	1111 1111 1111 1000	FFF8
-25	1111 0011 1000 0000	F380
-55	1110 0100 1000 0000	E480

<sup>(1)</sup> The last two bits are high impedance and are shown as 00 in the table.

Table 2. Temperature Data Format

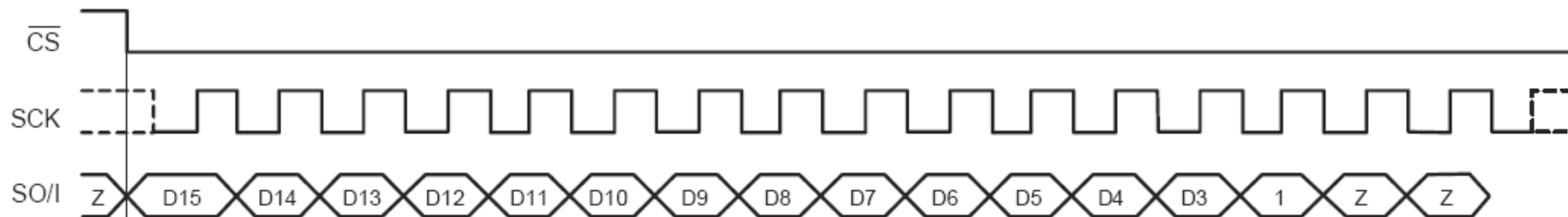


# Ramka SPI termometru TMP121



D15	D14	D13	D12	D11	D10	D9	D8
T12	T11	T10	T9	T8	T7	T6	T5
D7	D6	D5	D4	D3	D2	D1	D0
T4	T3	T2	T1	T0	0	Z	Z

Table 1. Temperature Register





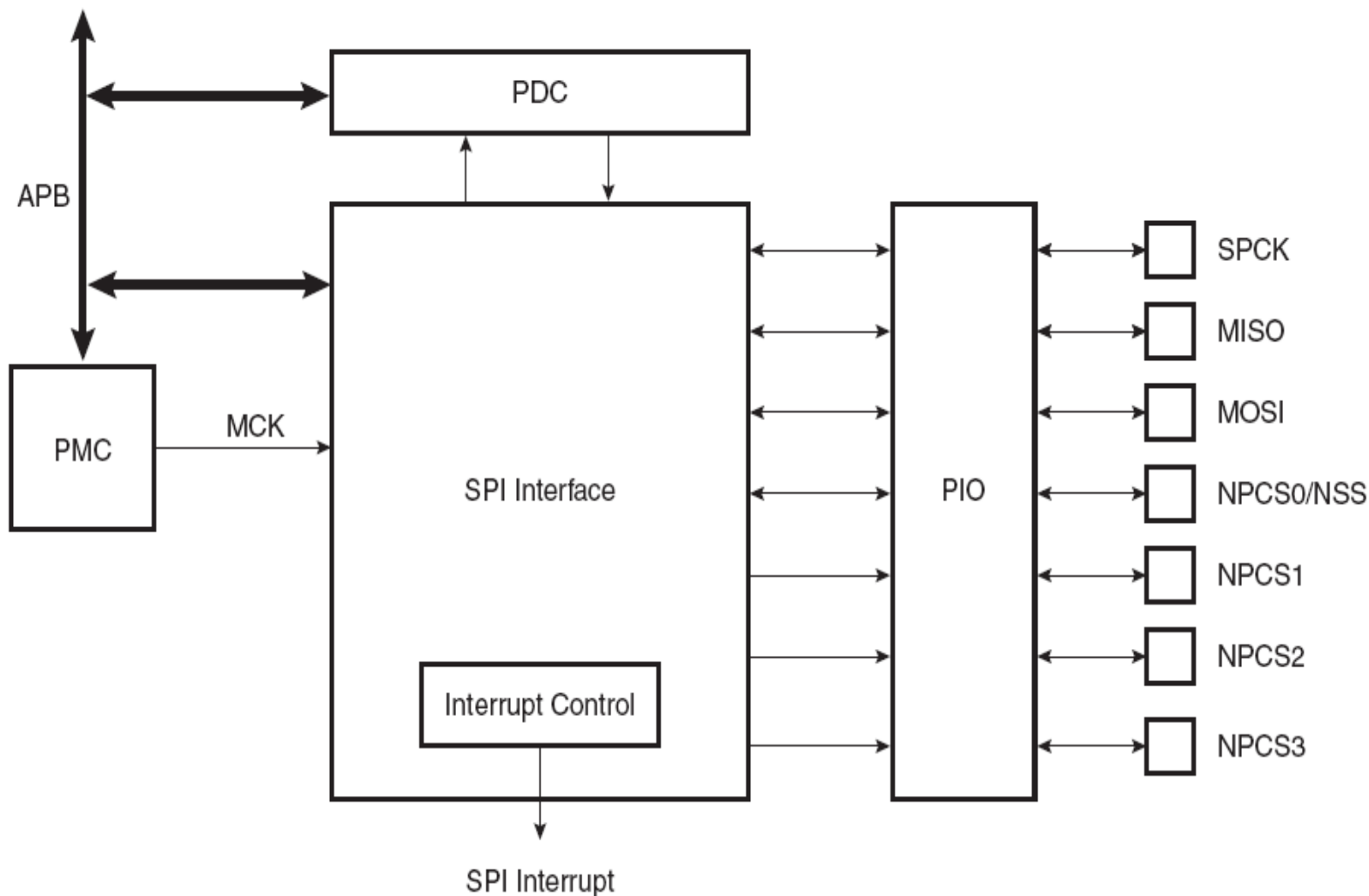
## Moduł SPI procesora ARM AT91SAM9263 (1)

### Cechy modułu SPI:

- Obsługa transferów w trybie Master lub Slave,
- Bufor nadawczy, odbiorczy oraz bufor transceivera,
- Transfery danych od 8 do 16 bitów,
- Cztery programowalne wyjścia aktywujące urządzenia dołączone do SPI (obsługa do 15 urządzeń),
- Programowalne opóźnienia pomiędzy transferami,
- Programowalna polaryzacja i faza zegara.

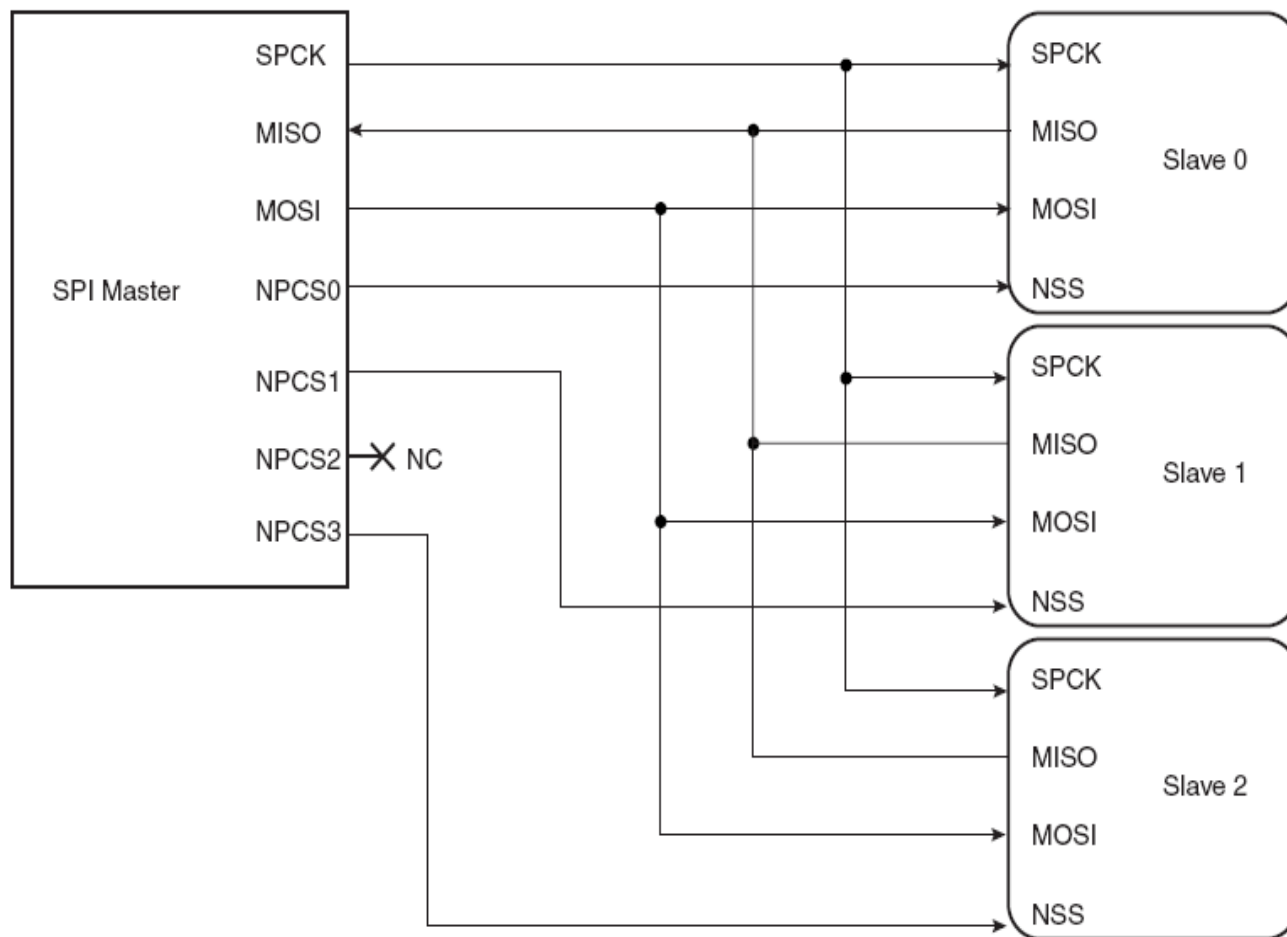


# Moduł SPI procesora ARM AT91SAM9263 (2)





# Moduł SPI procesora ARM (3)





- Informacje o zwolnieniu z egzaminu
- Hasło do pliku z informacją o zwolnieniu: sW2019



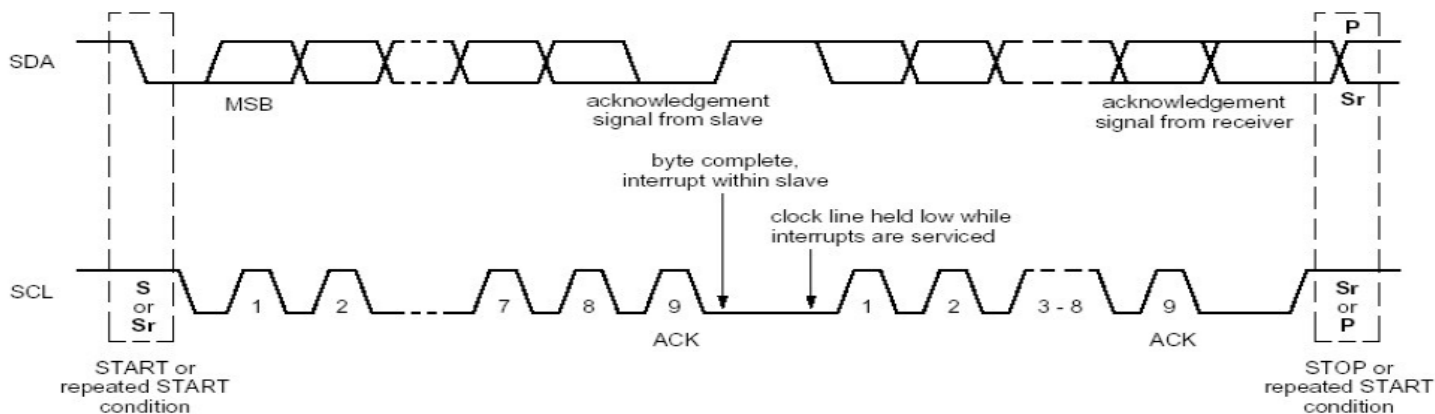


# Magistrala I2C



# Magistrala I2C

- Standard opracowany przez firmę Philips na początku lat 80,
- Dwuprzewodowy interfejs synchroniczny (SDA – linia danych, SCL – linia zegara),
- Transmisja dwukierunkowa, typu master-slave (multi-master), ramki 8-bitowe,
- Szybkość transmisji:
  - 100 kbps (standard mode),
  - 400 kbps (fast mode),
  - 3,4 Mbps (high-speed mode),
- Urządzenia posiadają niepowtarzalne adresy (7-bitów lub 10-bitów),
- Synchronizacja przy pomocy sygnału zegarowego umożliwia pracę urządzeń komunikujących się z różnymi szybkościami,
- Liczba urządzeń dołączonych do magistrali ograniczona jest pojemnością mag. (400 pF),
- Mechanizmy arbitrażu umożliwiające uniknięcie kolizji i utraty danych.

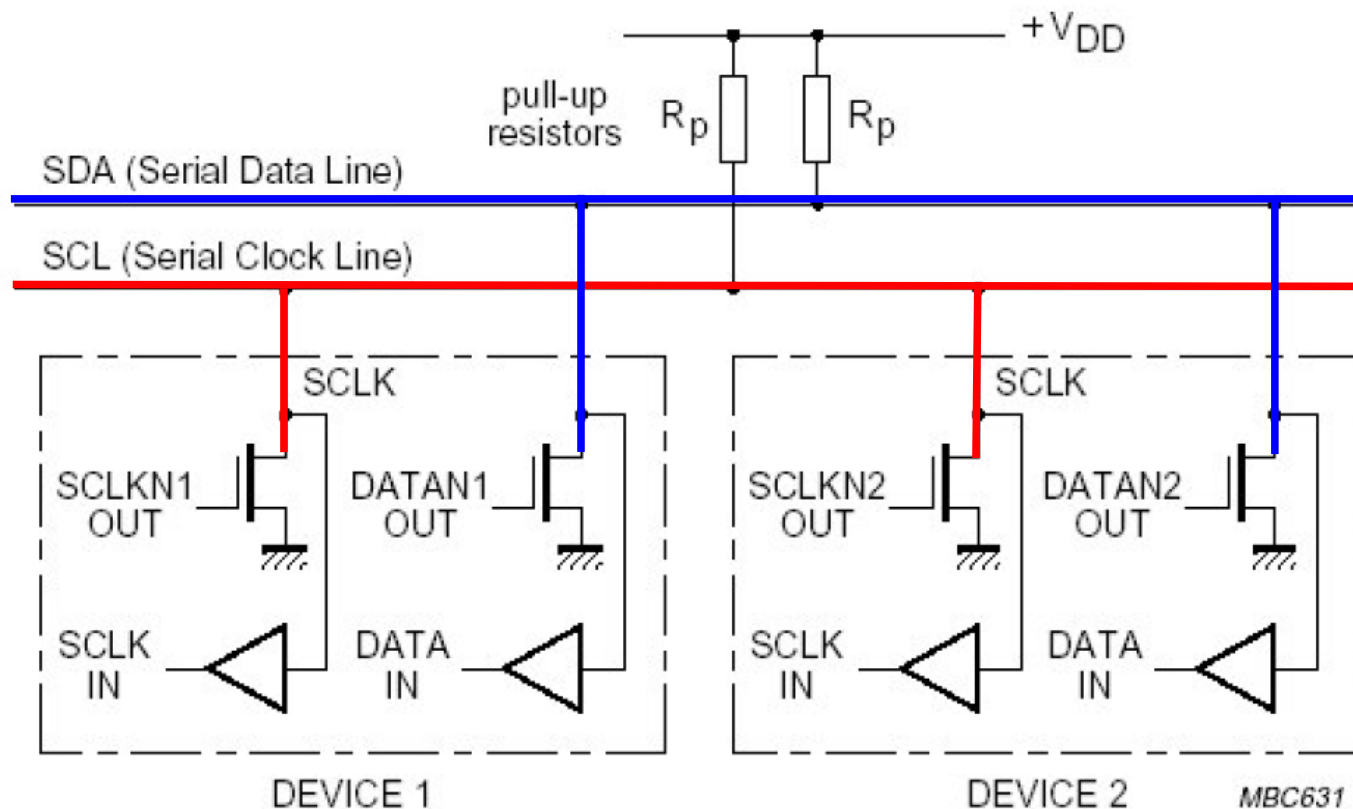




## Zastosowanie interfejsu I2C

W sprzedaży dostępnych jest wiele bardzo tanich układów scalonych sterowanych poprzez I<sup>2</sup>C:

- ★ PCF8563/8583 - zegar, kalendarz, alarm, timer, dodatkowo może służyć jako RAM
- ★ PCF8574 - pseudo-dwukierunkowy 8-bitowy ekspander
- ★ PCF8576, PCF8577 - sterowniki wyświetlaczy LCD
- ★ PCF8582 - pamięć EEPROM 256 bajtów (1, 2, 4 kB, ... MB)
- ★ PCF8591 - 8-bitowy, 4-kanalowy przetwornik analogowo-cyfrowy i cyfrowo-analogowy

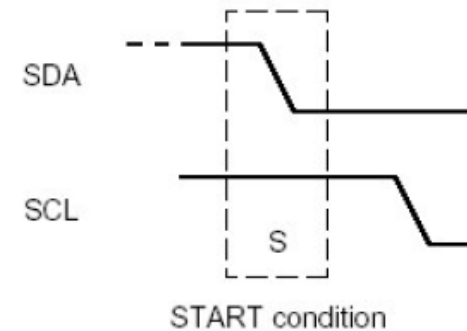


Urządzenie nadrzędne (Master) – inicjuje transmisję, generuje sygnał zegarowy

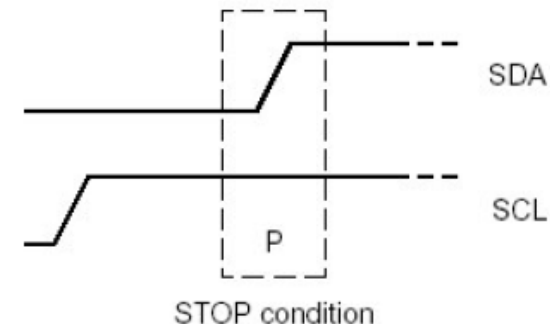
Urządzenie podrzędne (Slave) – analizuje wysłany przez urządzenie adres i transmituje lub odbiera dane.

## Rozpoczęcie oraz zakończenie transmisji

Rozpoczęcie transmisji – generacja sygnału **START** (opadające zbocze na szynie SDA, zmiana stanu z “1” na “0” logiczne, podczas ważnego sygnału SCL = ”1”). Sygnał generuje Master.

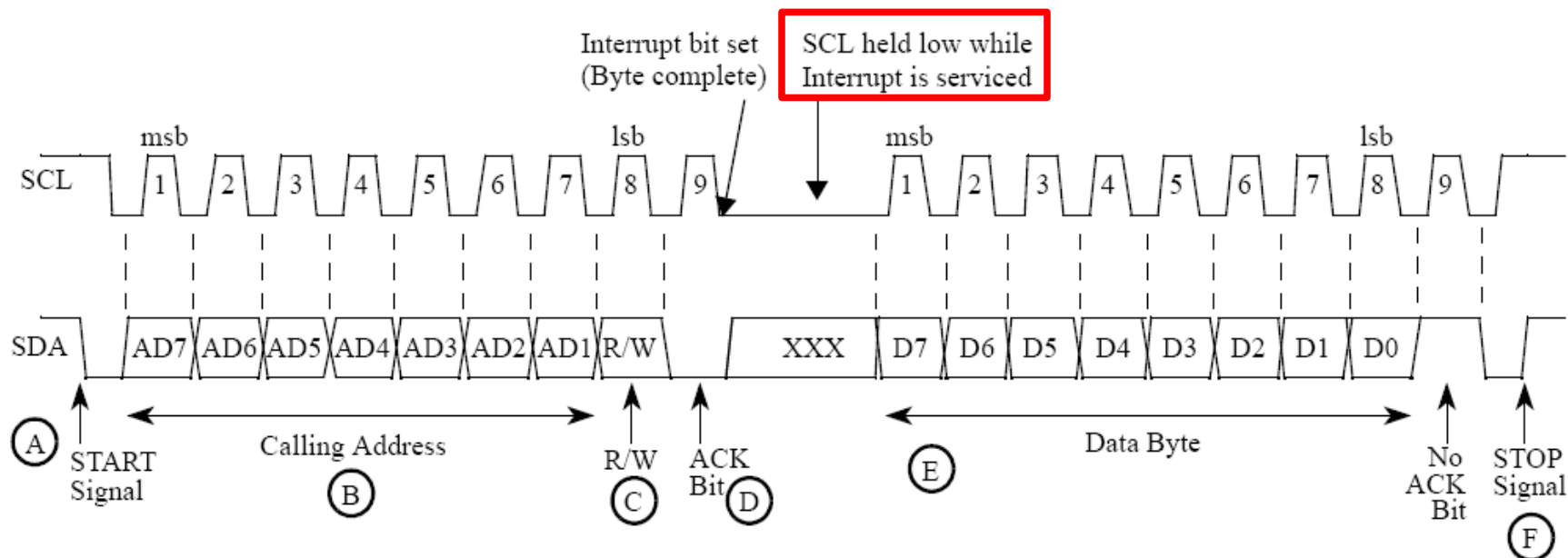


Zakończenie transmisji – generacja sygnału **STOP** (narastające zbocze na szynie SDA, zmiana stanu z “0” na “1” logiczną, podczas ważnego sygnału SCL = ”1”). Sygnał generuje Master.





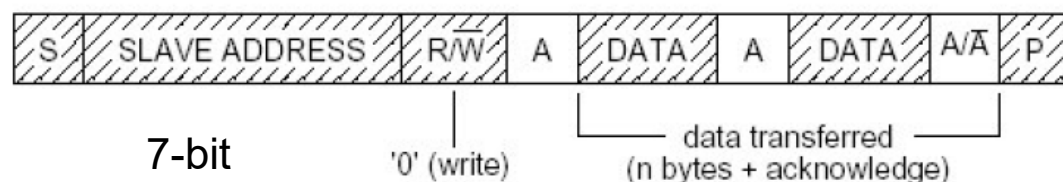
## Protokół I2C



- A) Transmisje rozpoczyna Master generując sygnał START.
- B) Następnie transmituje 8 bitów danych (7 bitów adresowych, bit R/W).
- C) Po transmisji 8 bitów Slave przejmuje magistralę i wymusza odpowiedni poziom na linii SDA (9 takt zegara). Odpowiada w ten sposób bitem potwierdzenia ACK (brak potwierdzenia, ACK = "1").
- E) Po przesłaniu adresu następuje faza odczytu lub zapisu danej do obsługiwanego urządzenia (8 bitów danych).
- F) Po przesłaniu danych urządzenie nadrzędne kończy transmisję generując brak potwierdzenia (ACK = "1") oraz bit stopu.



## Zapis n-bajtów danych master-transmitter



from master to slave

from slave to master

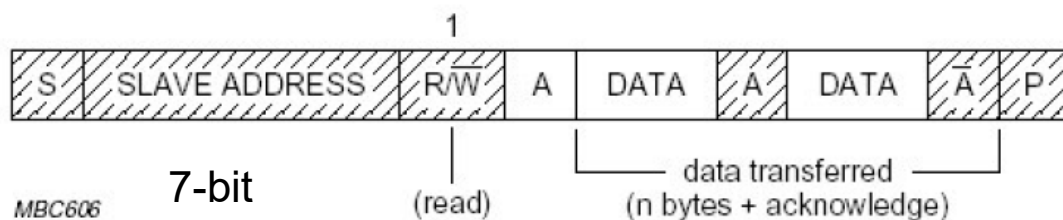
A = acknowledge (SDA LOW)

$\bar{A}$  = not acknowledge (SDA HIGH)

S = START condition

P = STOP condition

## Odczyt n-bajtów danych master-receiver (since second byte)



MBC606



## Two-Wire Interface – standard zgodny z I2C ?

Moduł TWI procesorów ARM jest odpowiednikiem standardu opracowanego przez firmę Philips (firma Philips posiada patent na interfejs I2C).

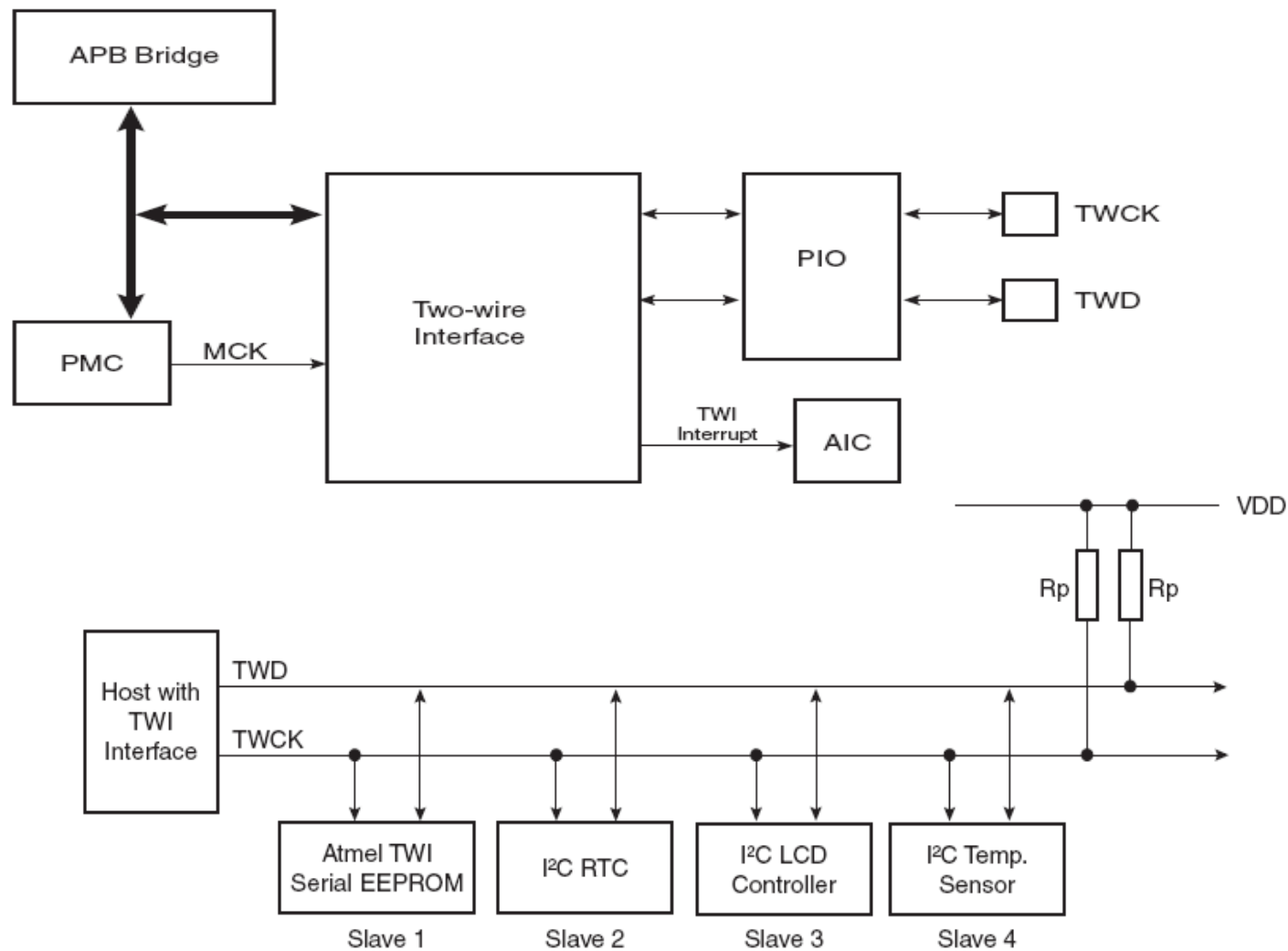
### Cechy interfejsu SWI procesora AMR firmy ATMEL:

- ★ Zgodny ze standardem I2C,
- ★ Praca w trybie Master, Multimaster lub Slave,
- ★ Umożliwia dołączenie urządzeń zasilanych napięciem 3,3 V,
- ★ Transmisja danych z częstotliwością zegara do 400 kHz,
- ★ Transfery poszczególnych bajtów wyzwalane przerwaniem,
- ★ Automatycznie przejście do trybu Slave w przypadku kolizji na magistrali (Arbitration-lost interrupt),
- ★ Przerwanie zgłaszane, gdy zostanie wykryty adres urządzenia w trybie Slave,
- ★ Automatyczne wykrywanie stanu zajętością magistrali,
- ★ Obsługa adresów 7 i 10-cio bitowych.





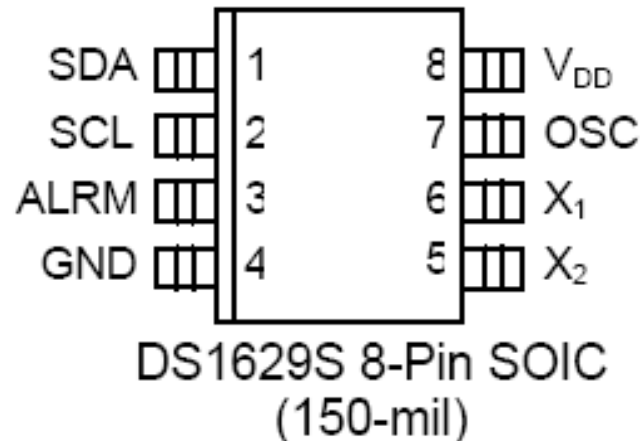
# Schemat blokowy modułu TWI





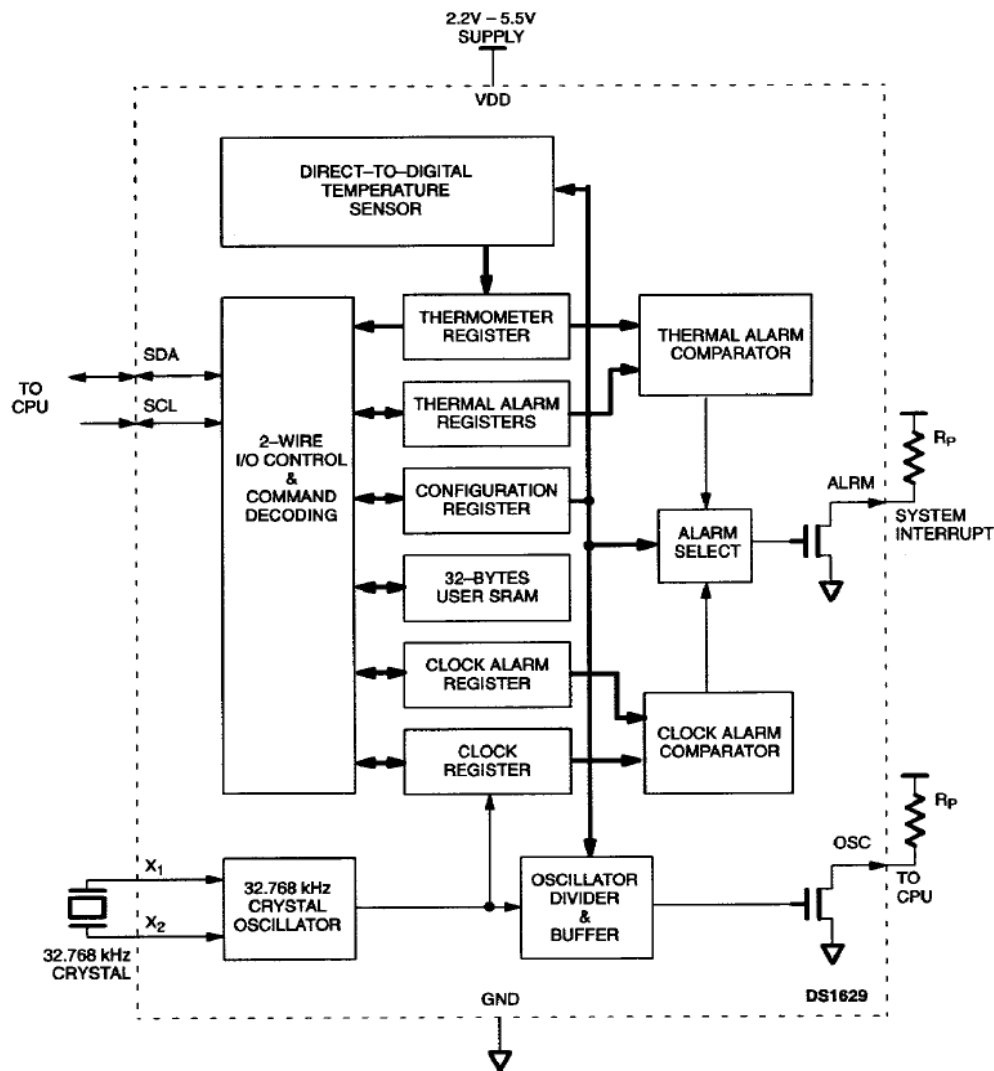
## Cechy układu DS1629:

- ★ Zegar czasu rzeczywistego,
- ★ Pomiar temperatury -55 – 125 C,
- ★ Rozdzielczość termometru: 9 bitów,
- ★ Dokładność termometru +/- 2 C,
- ★ Układ termostatu,
- ★ 32 bajty pamięci SRAM,
- ★ Zasilanie 2,2 – 5,5 V,
- ★ Interfejs zgodny ze standardem I2C (400 kHz).



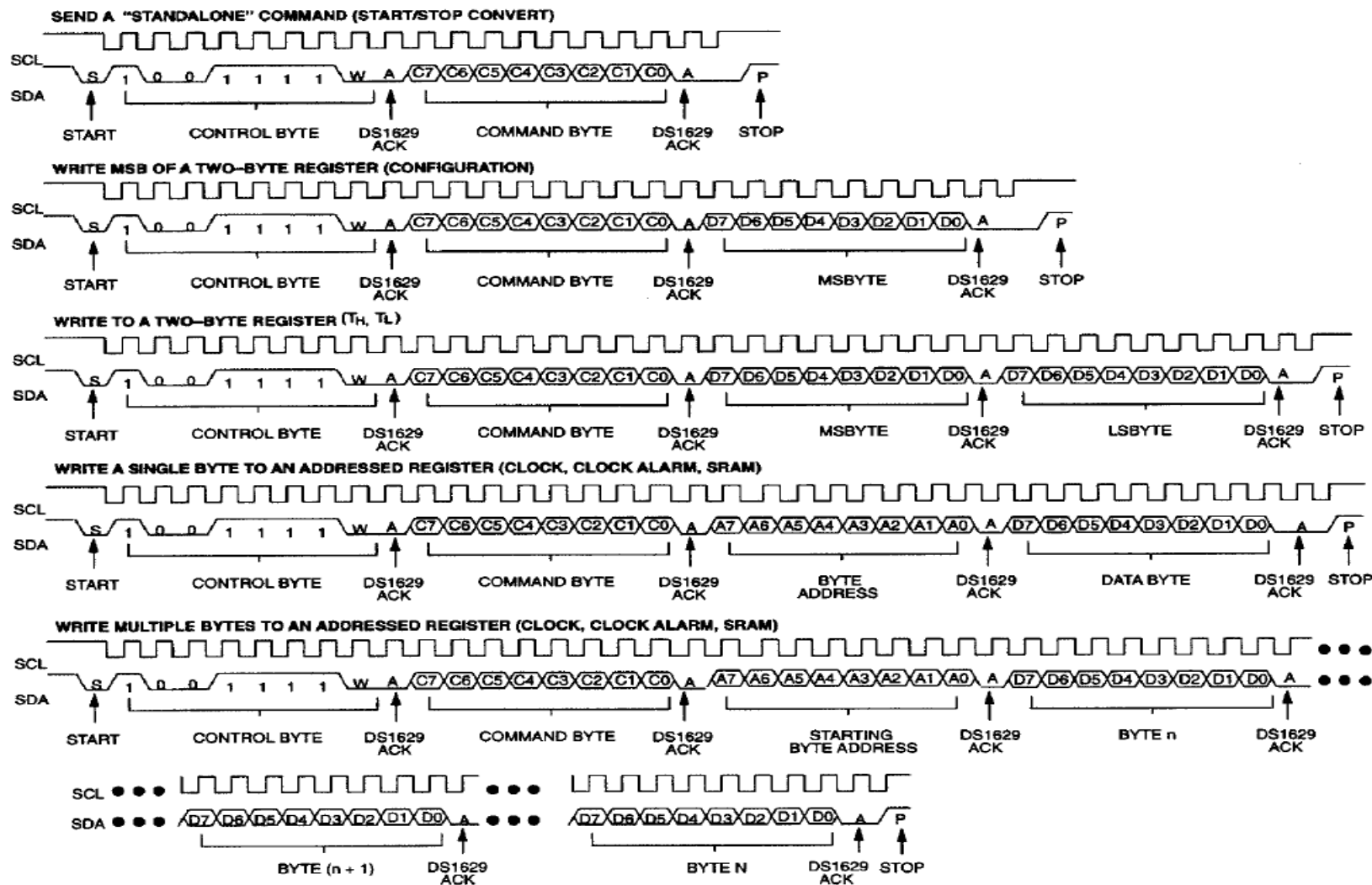


# Zegar czasu rzeczywistego



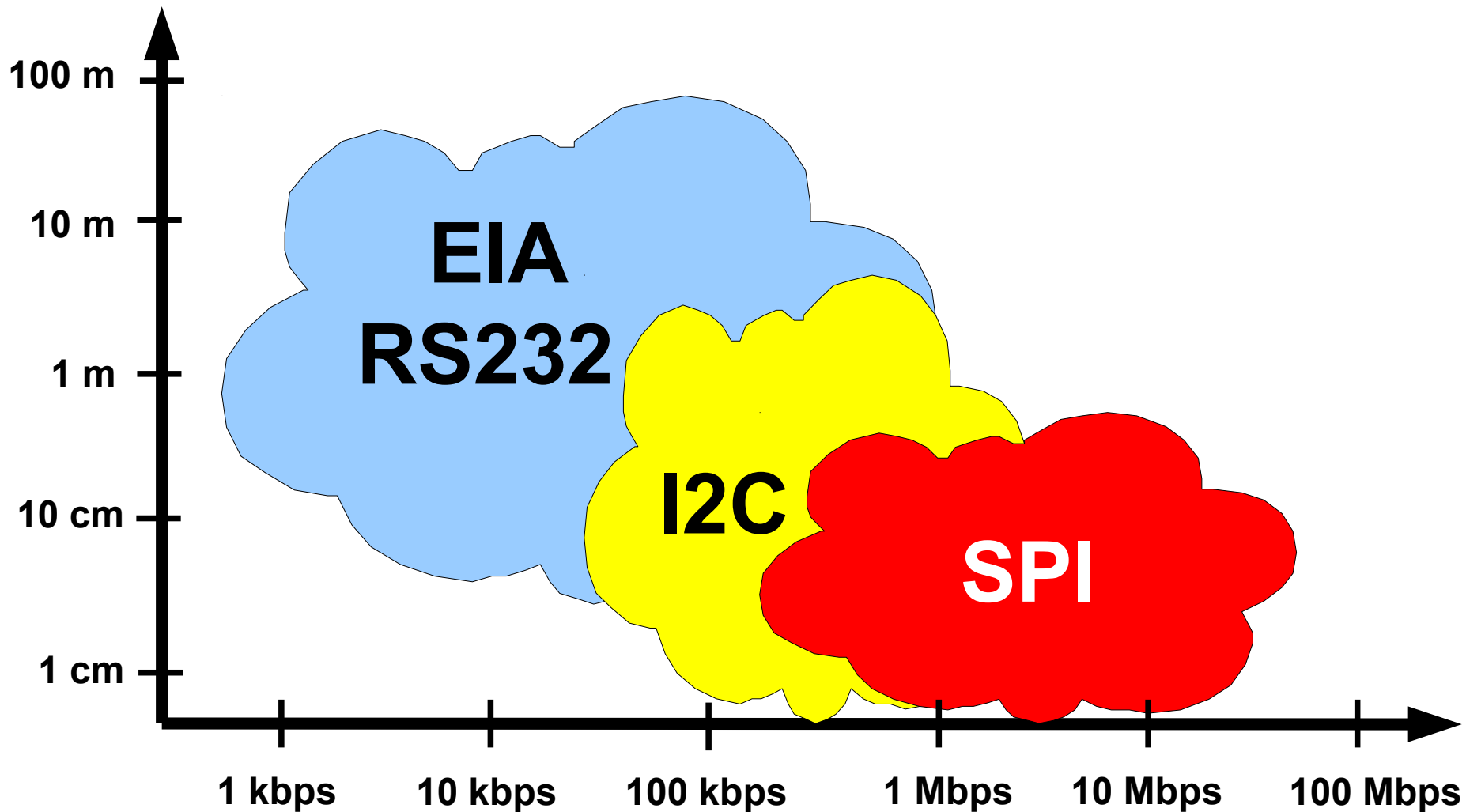


# Transmisja z wykorzystaniem interfejsu I2C





# Interfejsy szeregowo - podsumowanie

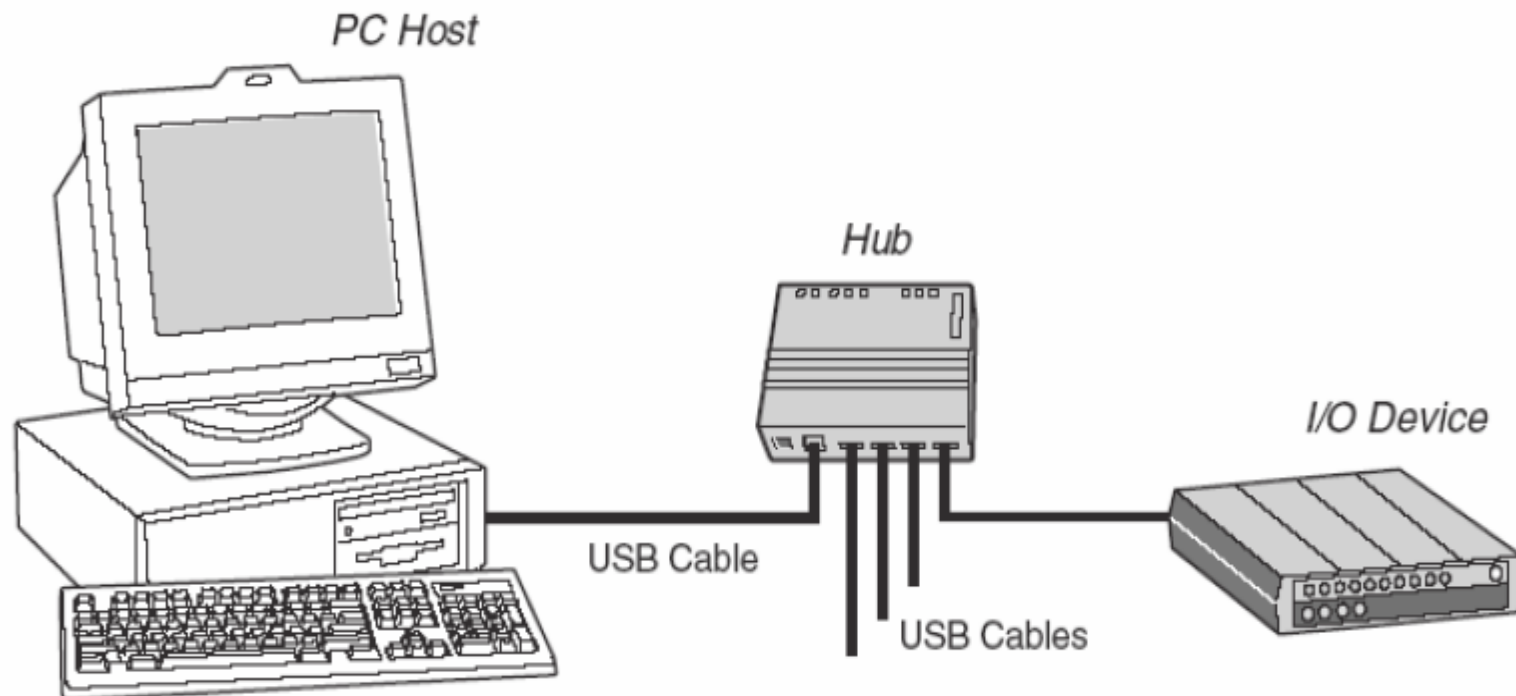




# Magistrala USB (Universal Serial Bus)



# Magistrala USB





## Cechy magistrali USB

- ★ Asynchroniczna, szeregowo, różnicowa transmisja danych,
- ★ Automatyczna detekcja dołączenia/odłączenia urządzenia oraz automatyczna konfiguracja,
- ★ Pojedyncze, ustandaryzowane złącze,
- ★ Możliwość dołączenia do 127 urządzeń do magistrali,
- ★ Automatyczna detekcja i korekcja błędów,
- ★ Szybkość transmisji danych:
  - ➔ LOW 1.5 Mb/s, specyfikacja USB >1.1,
  - ➔ FULL 12 Mb/s, specyfikacja USB >1.1,
  - ➔ HIGH 480 Mb/s, specyfikacja USB 2.0,
  - ➔ **Specyfikacja USB 3.0 => 5 Gb/s.**

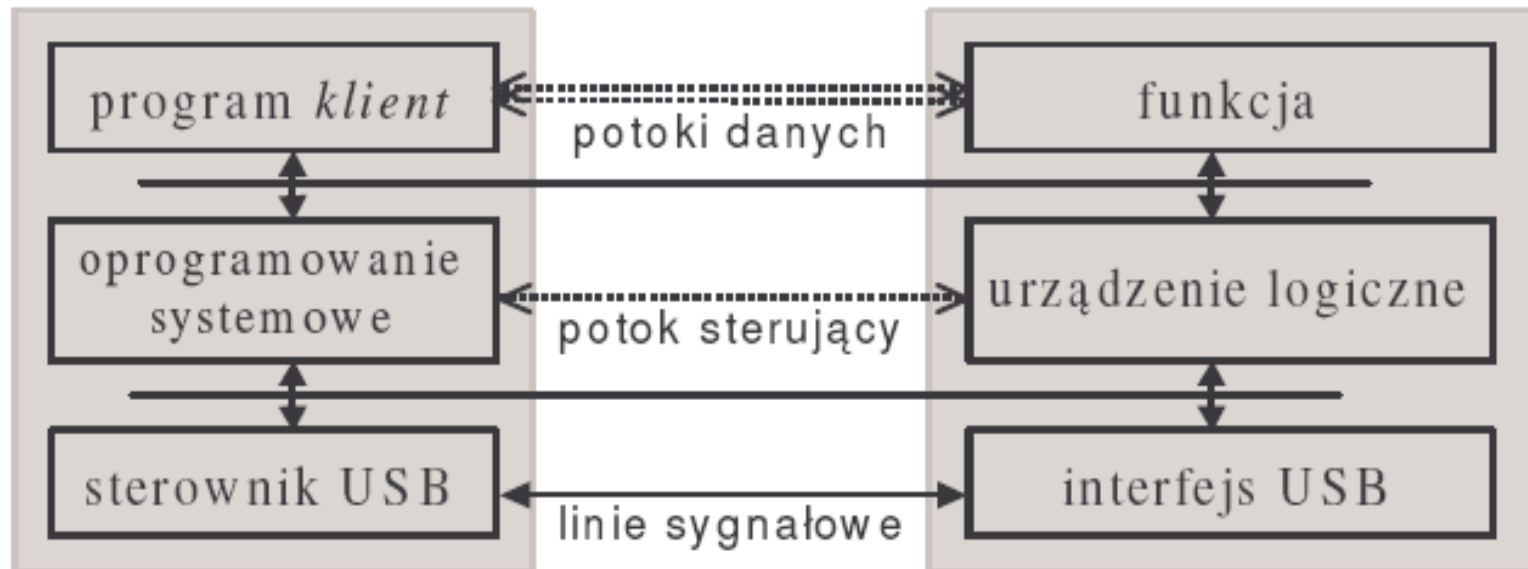
<u>TRANSMISJA</u>	<u>PRZYKŁADOWE ZASTOSOWANIA</u>	<u>CZĘSTOTLIWOŚĆ PRACY INTERFEJSU USB</u>
<b>WOLNA</b> 10 - 100 kb/s	Klawiatura, mysz, manipulATORY.	<b>mała - 1,5 Mb/s</b>
<b>ŚREDNIA</b> 500 kb/s - 10 Mb/s	Urządzenia do transmisji danych po liniach telefonicznych, urządzenia audio.	<b>pełna - 12 Mb/s</b>
<b>SZYBKA</b> 25 - 400 Mb/s	Urządzenia wideo, pamięci dyskowe.	<b>duża - 480 Mb/s</b>



## Struktura warstwowa magistrali USB

### Komputer macierzysty

### Urządzenie USB

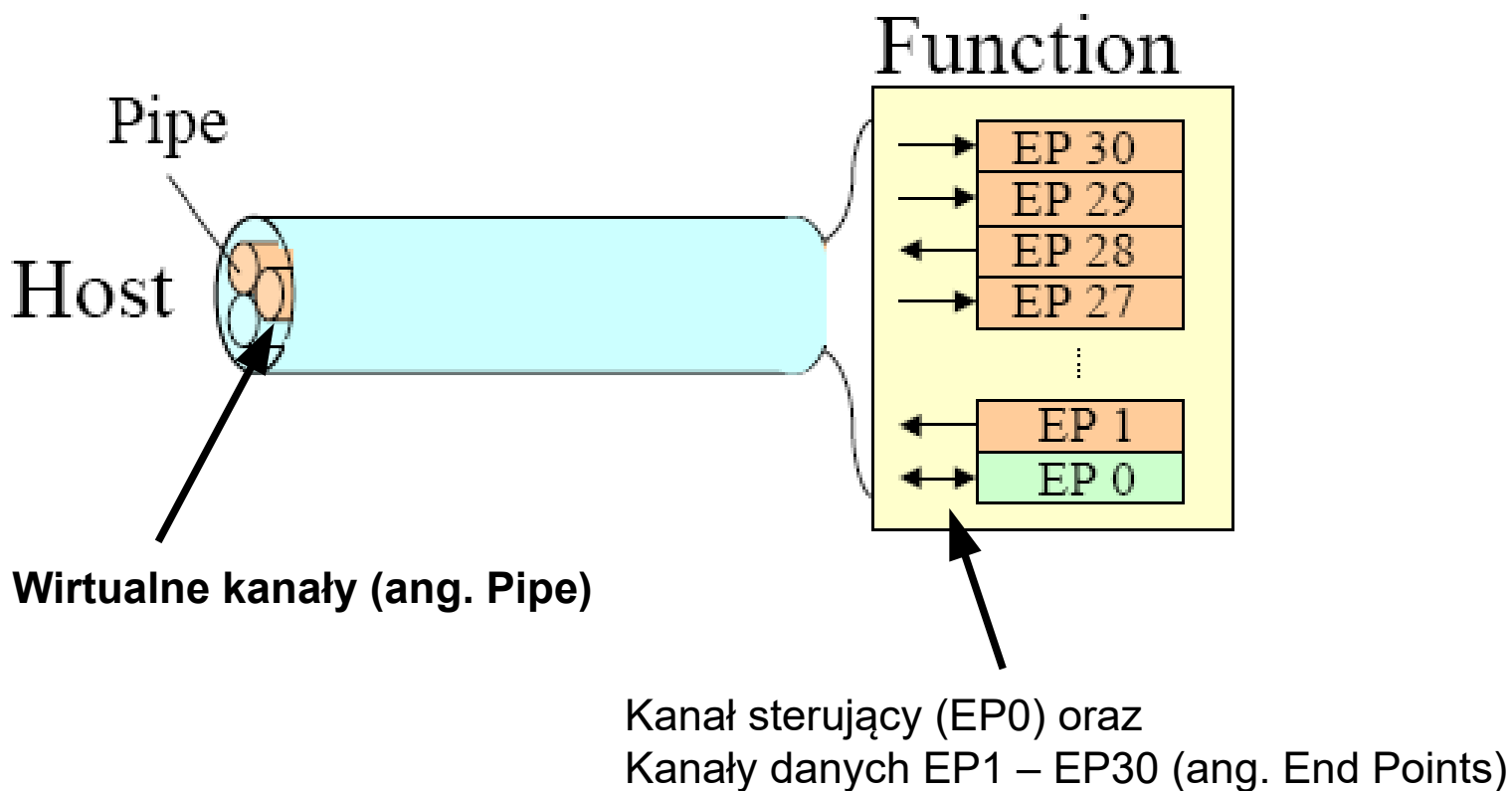


**Magistrala USB zbudowana jest na bazie architektury typu gwiazda.**

Model systemu USB składa się z trzech warstw:

- ◆ warstwa fizyczna,
- ◆ warstwa logiczna,
- ◆ warstwa funkcjonalna.

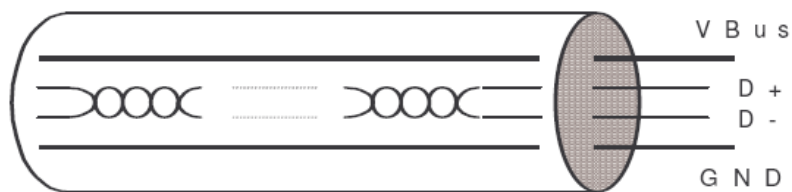
# Przepływ danych w systemie USB



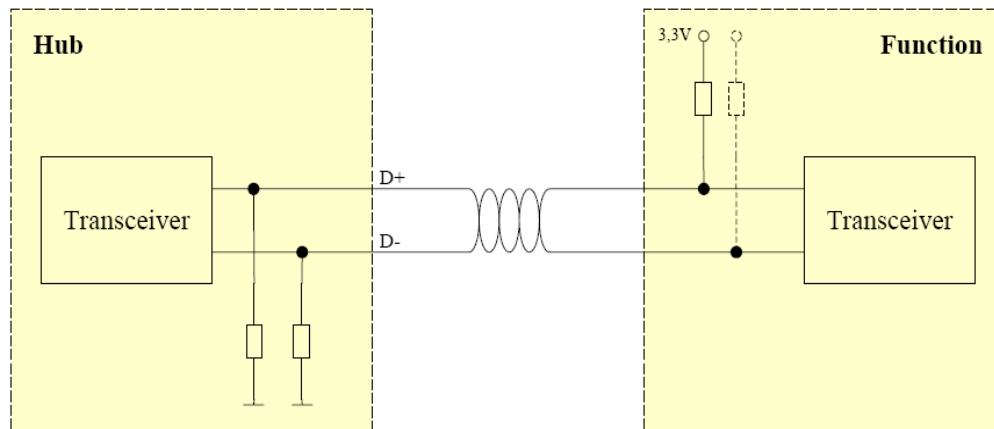


# Warstwa fizyczna

Maksymalnie 5 metrów



Transmisja różnicowa, typu half-duplex. Dwa dodatkowe przewody zasilające 5 V/500 mA



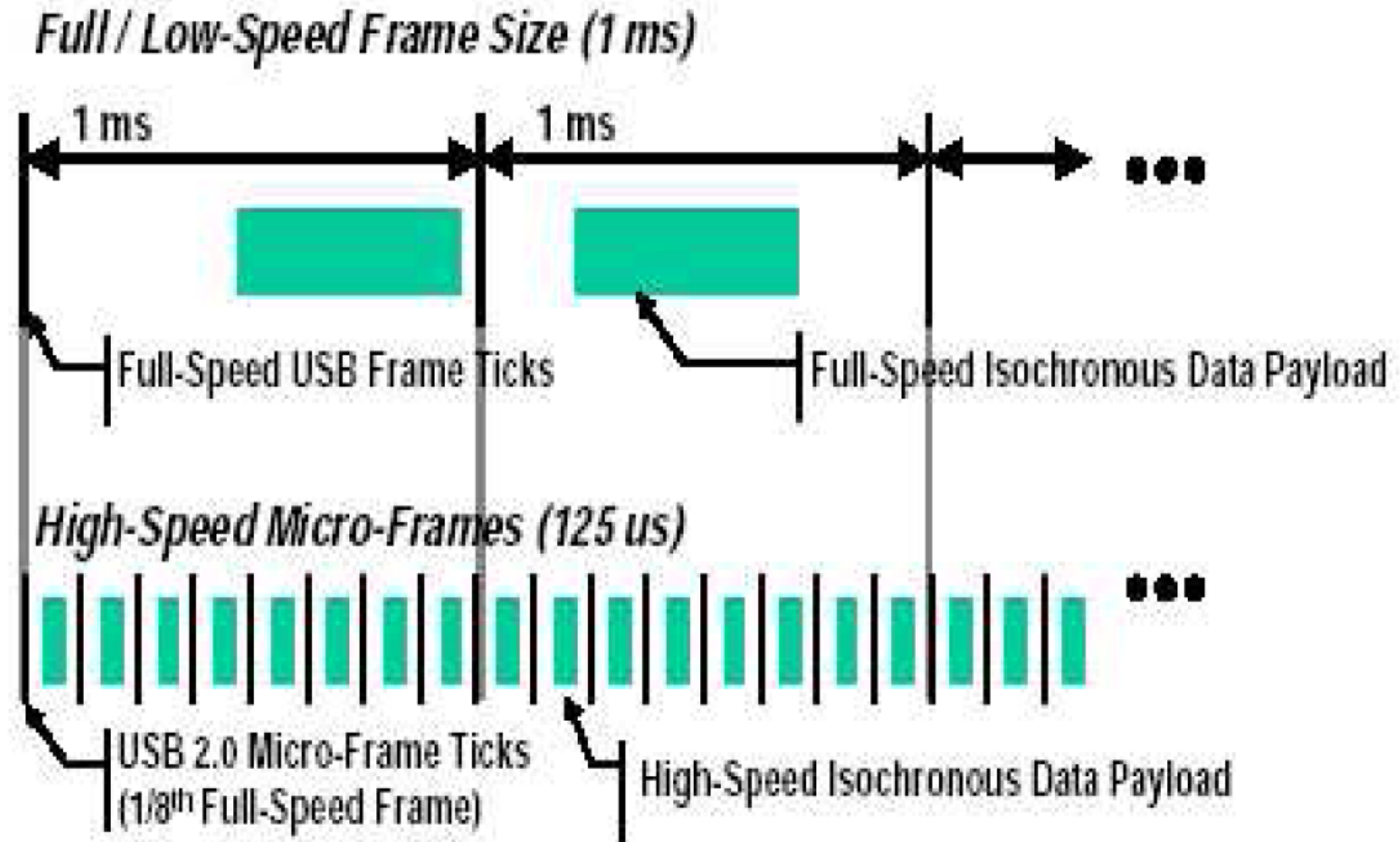
Złącza typu mini USB



Złącza USB typu "A" i "B"



## Zależności czasowe ramek USB





# Rodzaje transferów

Type	Important attributes	Max size LS	Max size FS	Max size HS	Examples
Interrupt	Quality + time	8	64	3072	Mouse, keyboard
Bulk	Quality	-	64	512	Printer, scanner
Isochronous	time	-	1023	3072	Audio, video
Control	Quality + time	8	64	64	System control



## Proces konfiguracji

**Enumeracja (ang. Enumeration)** – konfiguracja urządzeń przeprowadzana po dołączeniu lub odłączeniu nowego urządzenia od magistrali. Proces konfiguracji przeprowadzany jest przez urządzenie nadrzędne (Master). Master przypisuje indywidualne adresy do urządzeń oraz ustanawia podstawowe parametry transmisji:

- ★ Adres urządzenia w przestrzeni USB,
- ★ Rodzaj transferu,
- ★ Kierunek transmisji danych (read, write, read-write),
- ★ Rozmiar przesyłanych pakietów,
- ★ Szybkość transmisji,
- ★ Adresy buforów używanych przez sterowniki urządzenia,
- ★ Prąd pobierany przez urządzenie.



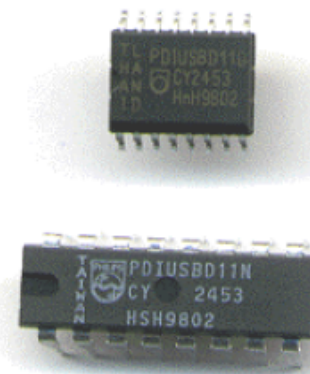
# Koncentratory USB



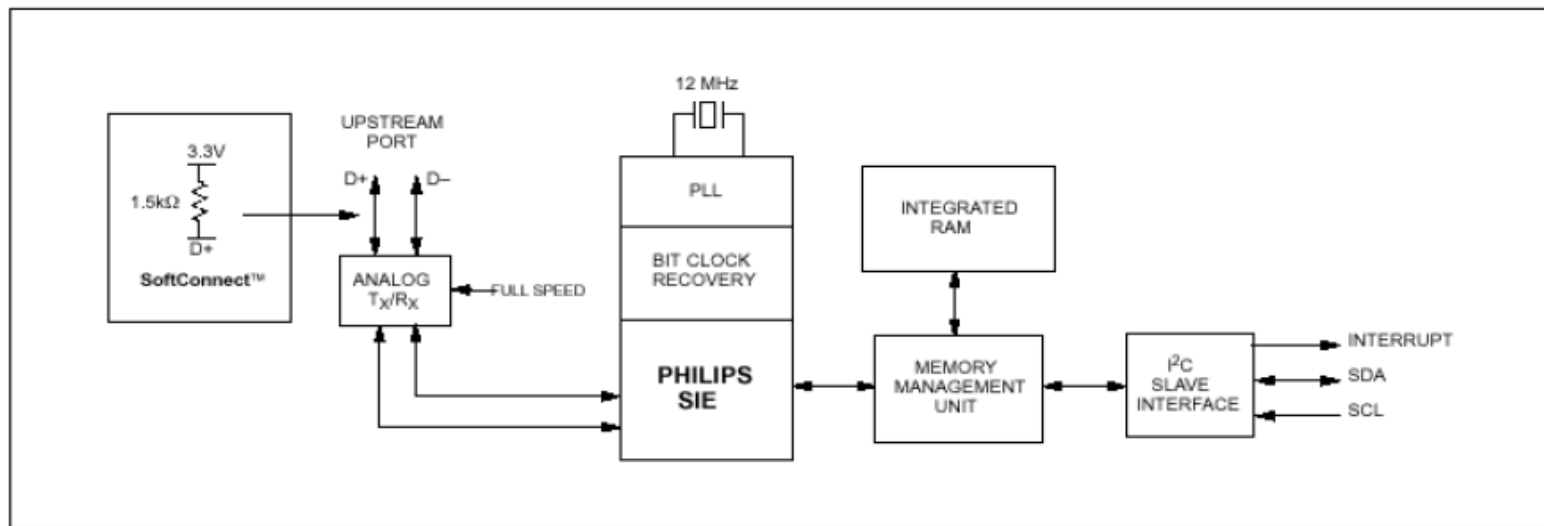


# Konwerter USB - I2C

## Philips PDIUSB011 (USB to I2C)



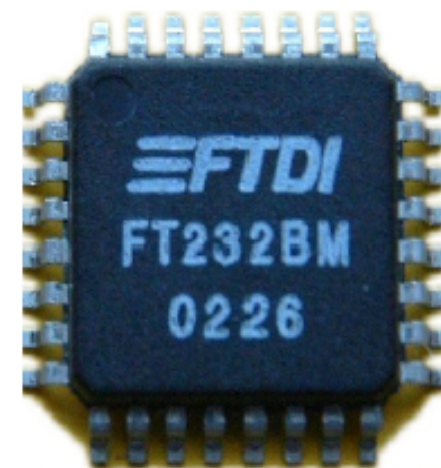
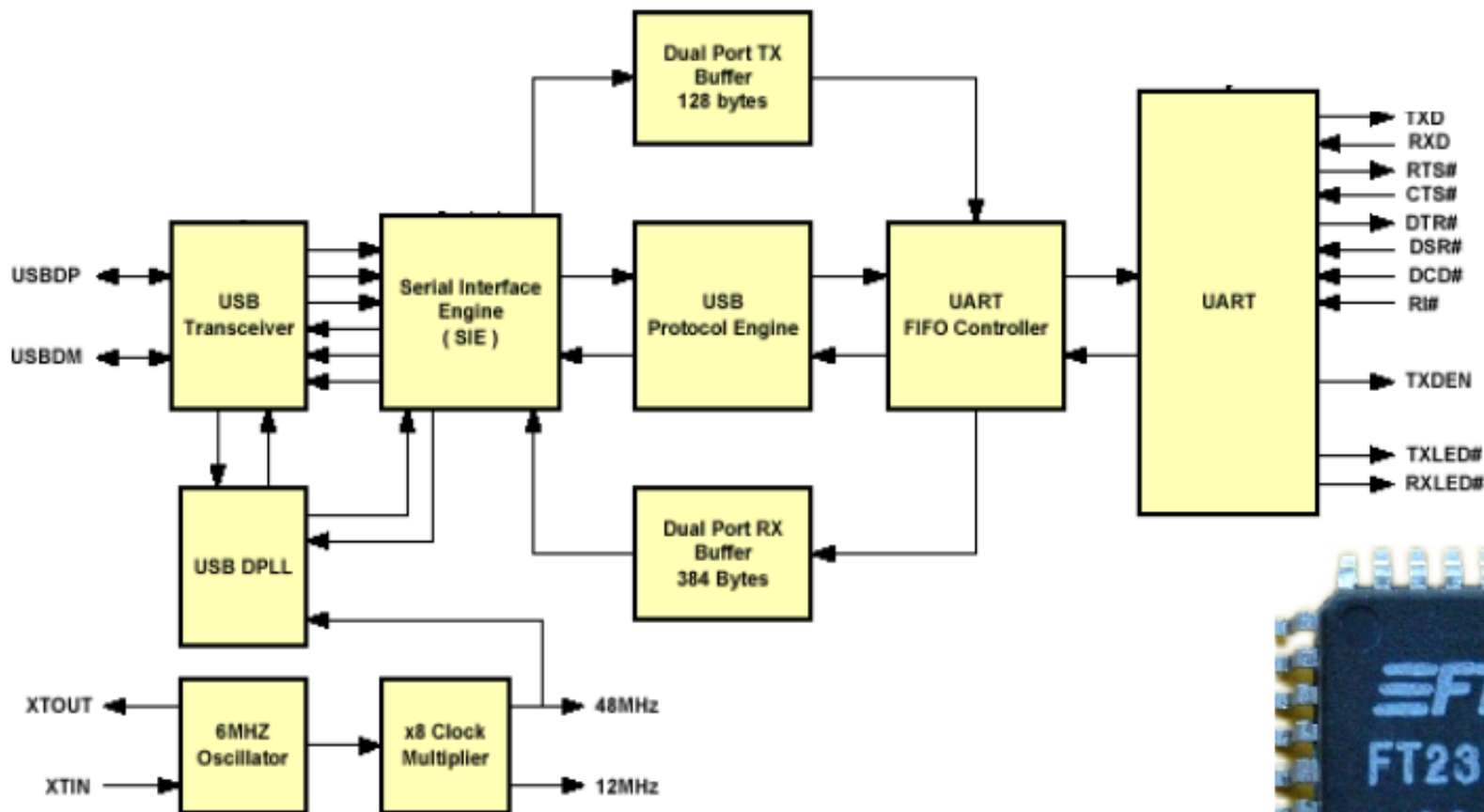
### BLOCK DIAGRAM







# Konwerter EIA 232-USB





## USB i procesory ColdFire a USB

### Low\Full speed:

MCF 527X (72-75)	66 – 166 MHz
MCF 5221X (72-75)	80 MHz
MCF 5222X (72-75)	80 MHz
MCF 527X (72-73)	240 MHz

68HCS08JW32            8 MHz

### High Speed:

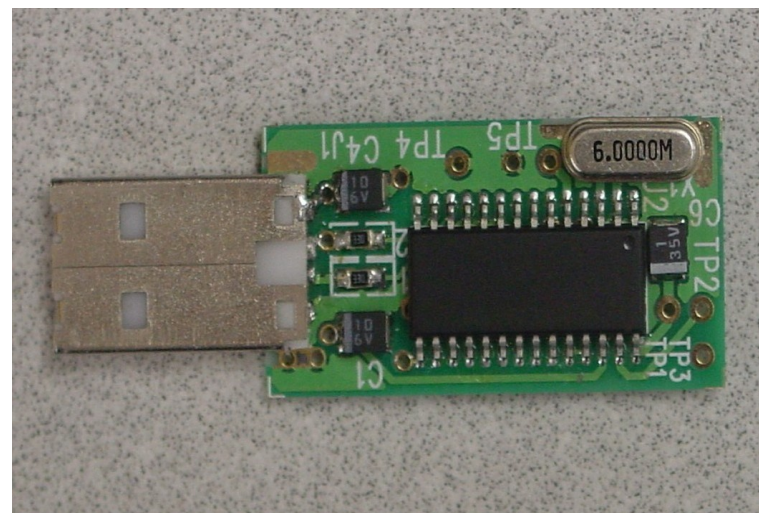
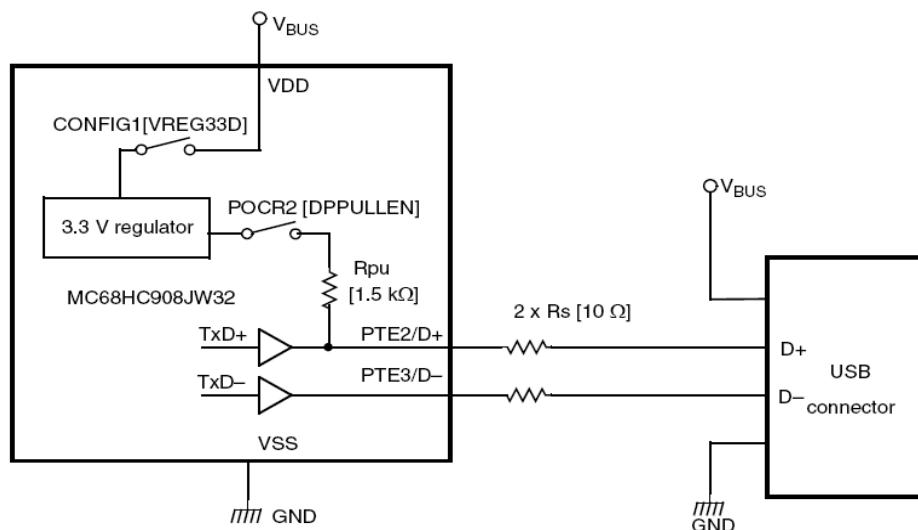
MCF 547X (72-75)	200 – 266 MHz
MCF 548X (82-85)	166 – 200 MHz
MCF 537X (77-79)	240 Mhz
MCF 5253	140 MHz



# Motorola 68HC908JW32

Cechy modułu USB procesora HC908:

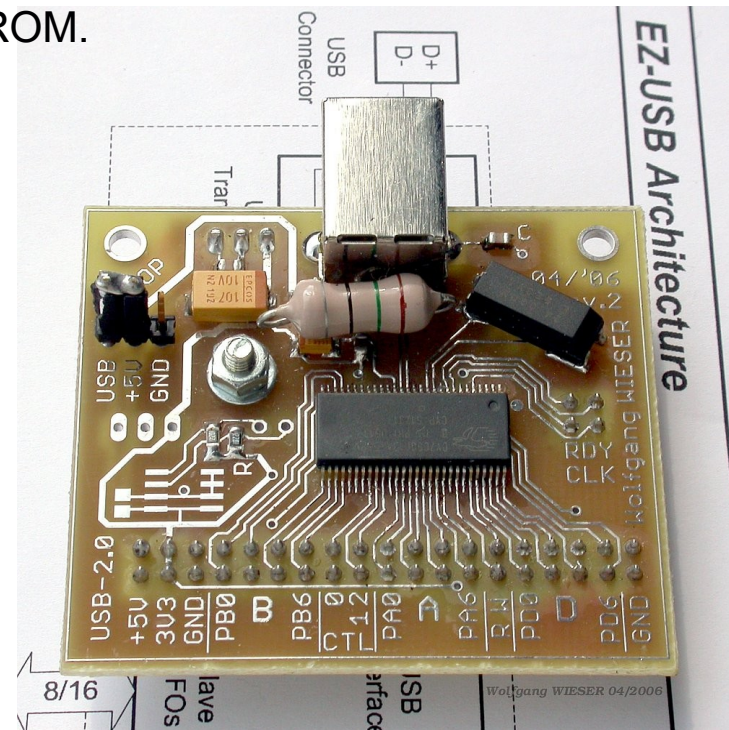
- Interfejs zgodny ze standardem USB 2.0 full speed,
- 12 Mbps data rate,
- Wbudowany stabilizator napięcia 3.3 V,
- Endpoint 0 wyposażony w 8-bytowy bufor nadawczy i odbiorczy
- 64 bajtowy bufor endpoint współdzielony przez bufory końcowe 1-4.



# Procesor Cypress CY7C68013A

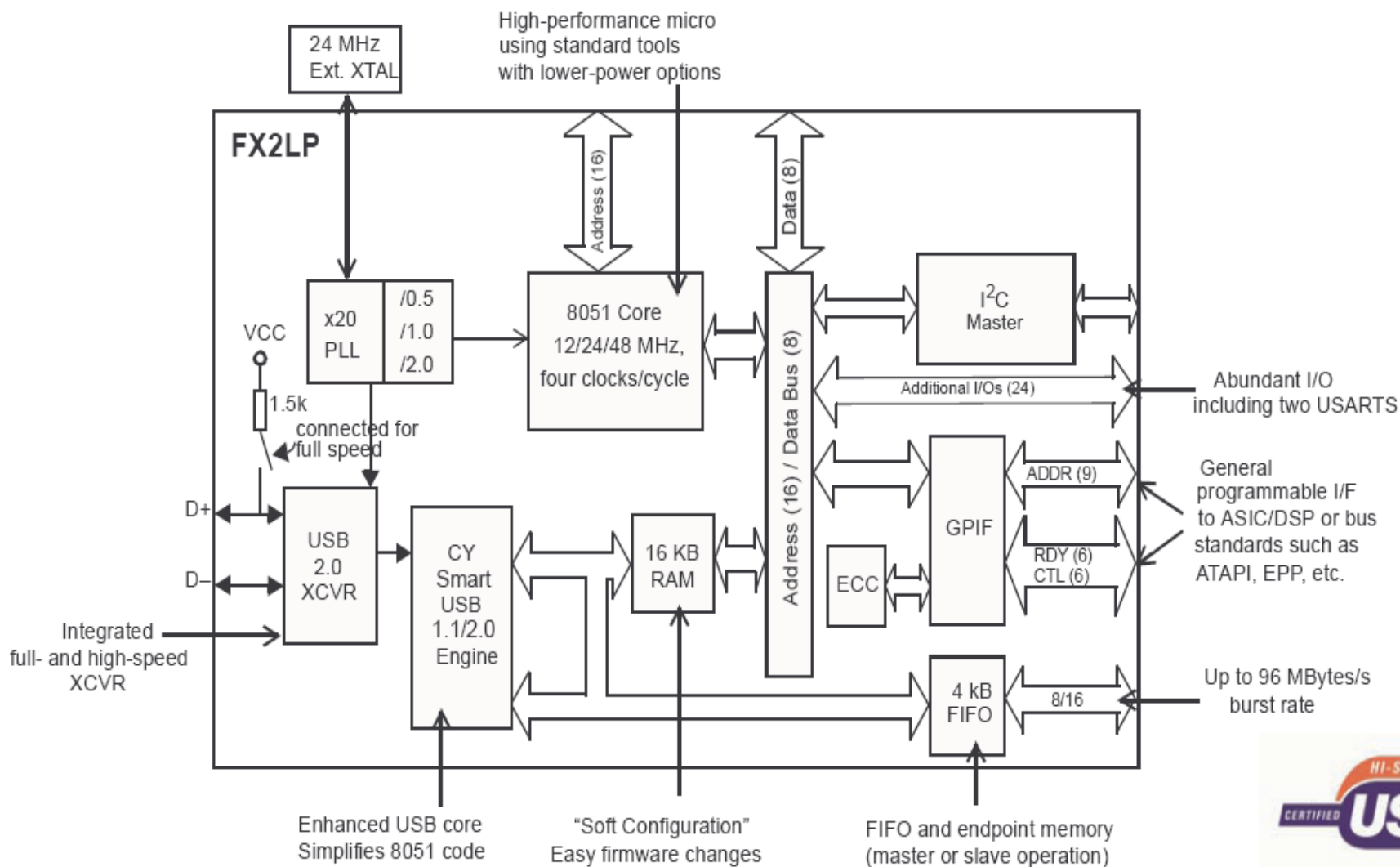
## Cechy procesora CY7C68013:

- ★ Interfejs zgodny ze standardem USB 2.0–USB-IF high speed,
- ★ Rozbudowane jądro procesora rodziny **8051**,
- ★ Zintegrowana pamięć programu 16 kB (RAM)
  - ➔ Pamięć ładowana z USB,
  - ➔ Pamięć ładowana z zewnętrznej pamięci EEPROM.
- ★ Cztery programowalne bufory końcowe (BULK/INTERRUPT/ISOCRONOUS)
- ★ Dodatkowy 64 bajtowy endpoint (BULK/INTERRUPT),
- ★ 8- lub 16-bitowy interfejs zewnętrzny,
- ★ Kanał DMA, GPIF (General Programmable Interface)





# Procesor Cypress CY7C68013A





# USB 3.0

---

SuperSpeed USB 3.0 Specification  
Revolutionizes An Established Standard

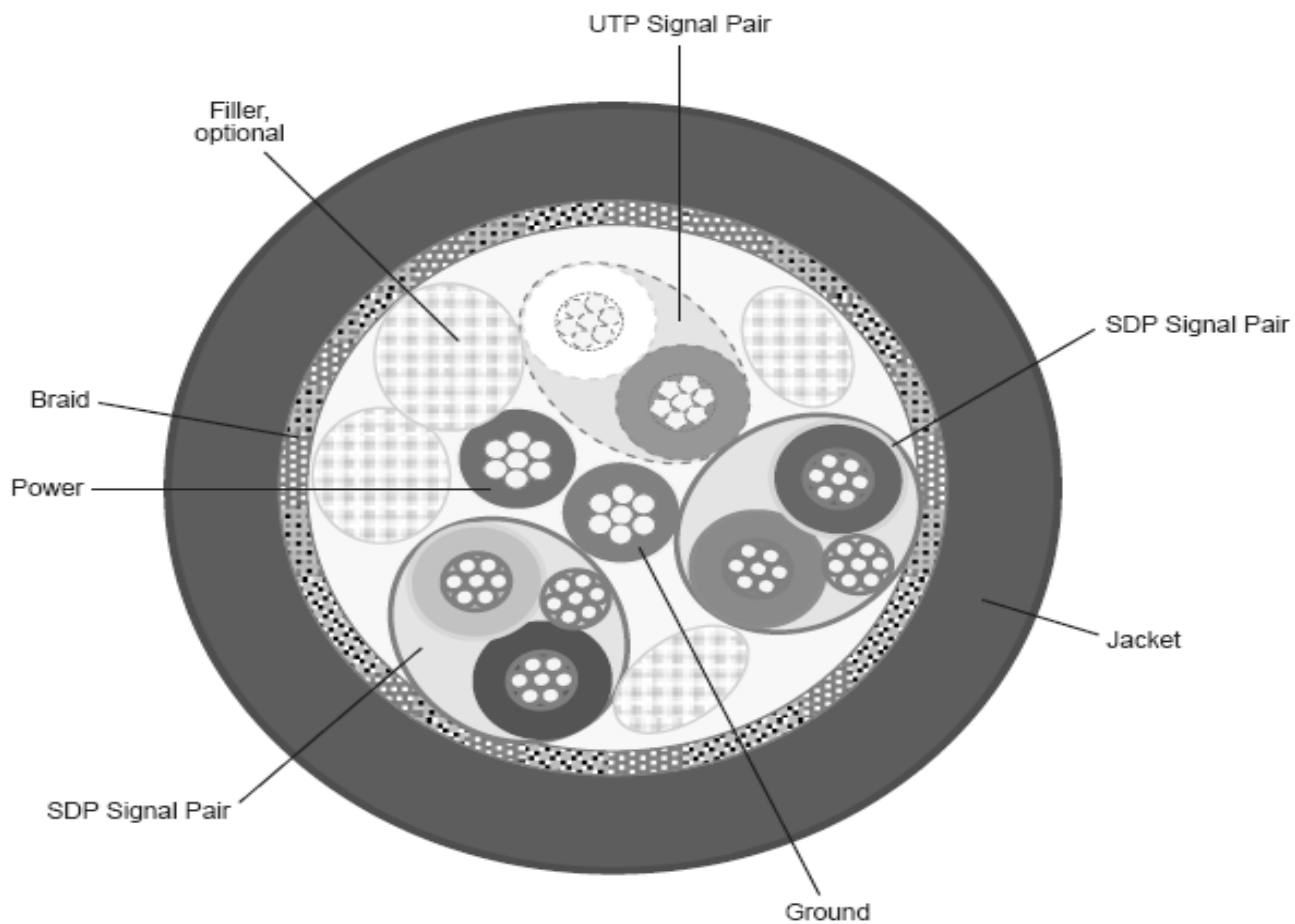




- Interfejs szeregowy, full-duplex
- Szybkość transmisji danych: 5 Gb/s (10 razy szybciej niż USB 2.0)
- Standard kompatybilny z USB 2.0 (sterowniki i złącza), jednak znacznie różniący się od USB 2.0
- Transmisja danych full-duplex, zasilanie
- Inteligentne zarządzanie poborem energii, mniejsze zużycie energii
- Warstwa łącza danych i fizyczna podobna do interfejsu PCI express 2.0



# Warstwa fizyczna USB 3.0



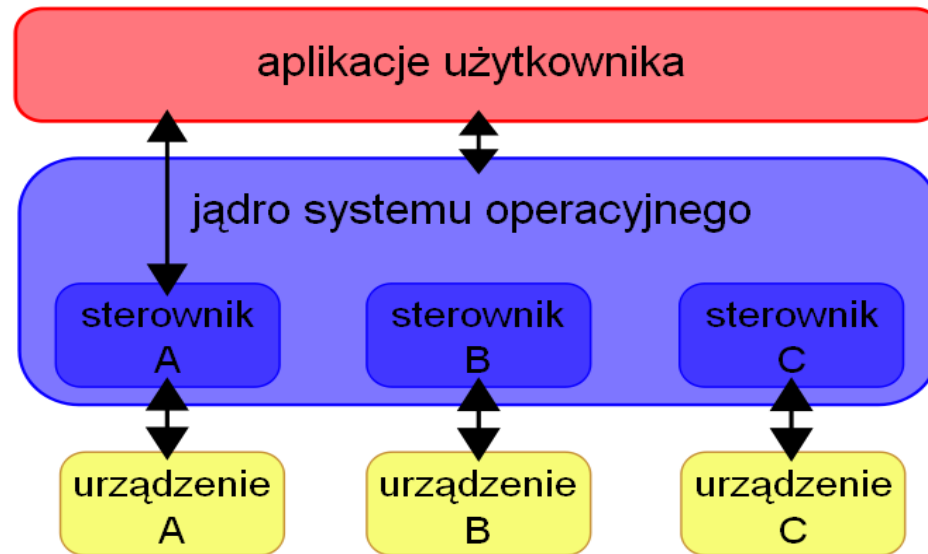




# Sterowniki urządzeń peryferyjnych



# Sterowniki urządzeń (1)



**Sterownik urządzenia (ang. driver)** - program lub fragment programu odpowiadający za dane urządzenie i pośredniczący pomiędzy nim, a resztą systemu komputerowego. Sterownik zwykle traktowany jest jako zestaw funkcji przeznaczonych do obsługi urządzenia peryferyjnego. Sterownik odwzorowuje pewne cechy urządzenia. Nazewnictwo funkcji oraz parametry przyjmowane i zwracane przez funkcje są zwykle narzucone przez system operacyjny. Sterowniki urządzeń dostępne w systemach operacyjnych udostępniają programiście interfejs API (Application Programming Interface), **bezpośredni dostęp do urządzenia jest zabroniony.**

W przypadku systemów wbudowanych **aplikacje mogą się bezpośrednio odwoływać do urządzeń**, czasami trudno jest odróżnić aplikację od sterownika.

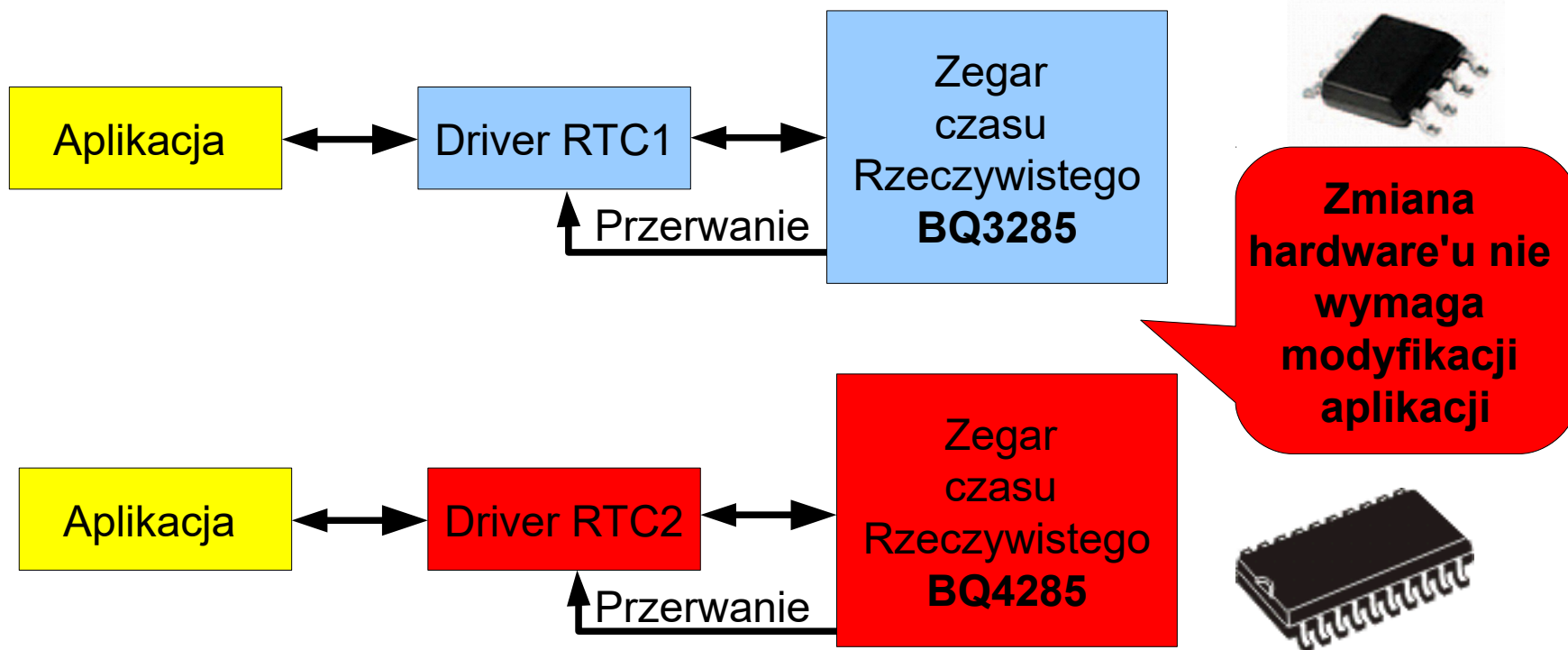


## Sterowniki urządzeń (2)

**Sterownik urządzenia peryferyjnego** (urządzenia peryferyjne wewnętrzne i zewnętrzne) udostępnia podstawowe funkcje pozwalające na łatwe korzystanie z danego urządzenia.

Sterownik urządzenia pozwala programiście „ukryć” dane urządzenie – dostarczając tylko zestaw funkcji umożliwiających sterowanie oraz wymianę danych z danym urządzeniem.

Pisząc sterownik urządzenia musimy pamiętać o procedurach obsługujących przerwania.





## Sterowniki – inicjalizacja urządzeń

**Sterownik urządzenia peryferyjnego** - zwykle implementowane są następujące funkcje:

**Device\_Open ()** - funkcja wykorzystywana do inicjalizacji urządzenia.

Funkcja może przyjmować parametry jeżeli sterownik obsługuje więcej niż jedno urządzenie, np. dwa porty USART, których rejestry są dostępne pod innymi adresami bazowymi.

Funkcja może zwrócić wynik operacji związanej z inicjalizacją urządzenia lub deskryptor (wskaźnik) dający dostęp do danego urządzenia (do rejestrów urządzenia lub struktury umożliwiającej komunikację z nim).

Sterownik urządzenia może zostać „otworzony” przez kilka różnych aplikacji.

W takim przypadku należy zaimplementować, tzw. licznik odwołań do urządzenia. Licznik odwołań zwiększany jest przy każdym wywołaniu funkcji Open. Inicjalizacja urządzenia przeprowadzana jest tylko jeden raz.

**Device\_Close ()** - funkcja wywoływana, gdy aplikacja przestaje korzystać z danego urządzenia. Zadaniem funkcji jest bezpieczne wyłączenie urządzenia, np. w przypadku portów IO – konfiguracja jako porty wejściowe, USART – wyłączenie nadajnika/odbiornika, zamaskowanie przerwań.

Jeżeli funkcja Open została wykonana kilka razy, z urządzenia korzysta kilka aplikacji, należy jedynie zdekrementować licznik odwołań. Urządzenie wyłączane jest w przypadku, gdy licznik odwołań zmniejszy się do 0. Podobnie do funkcji Open, funkcja może przyjmować parametry oraz zwracać rezultat operacji.

Funkcje Open i Close powinny również konfigurować przerwania skojarzone z danym urządzeniem peryferyjnym.



## Sterowniki – komunikacja z urządzeniami

**ReadData Device\_Read ()** - funkcja wykorzystywana do odczytywania danych z urządzenia, np. portu szeregowego. Funkcja do odczytu danych może być funkcją blokującą lub nie. Funkcja blokująca czeka, aż dane będą dostępne (możliwe jest wcześniejsze opuszczenie funkcji jeżeli upłynie określony okres czasu, a danych nadal nie ma - **timeout**). Timeout jest zwykle obliczany przez jeden z timerów procesora. W takim przypadku procesor czekając na nadejście danych może wykonywać inne obliczenia.

Funkcja Read może również korzystać z przerwania lub kanału DMA (Direct Memory Access). W takim przypadku dane wpisywane są do bufora. Gdy zgromadzi się odpowiednio duża ilość danych ustawiana jest flaga informująca o ich nadejściu lub zgłaszane jest przerwanie systemowe.

**Device\_Write ()** - funkcja wykorzystywana do zapisywania danych do urządzenia, np. do portu szeregowego. Funkcja do odczytu danych może być funkcją blokującą lub nie. Funkcja blokująca czeka, aż dane zostaną wysłane. Transmisja danych przez port szeregowy również zajmuje dużo czasu (przesłanie 1 znaku z szybkością 9600 bit/s zajmuje około 1 ms). W takim przypadku dane zgromadzone w buforze mogą być przesyłane przy wykorzystaniu przerwania – funkcja nieblokująca. Funkcja ustawia flagę informującą o zakończeniu transmisji. Wykorzystanie kanału DMA znacznie przyspiesza wykonanie operacji.

Funkcje mogą zwracać rezultat wykonane operacji, np. przesłanie danych przez port USART wymaga potwierdzenia poprawności ich odbioru. W takim przypadku po wysłaniu danych uruchamiany jest odbiornik, który czeka na przesłanie potwierdzenia zgodnego w użytym protokołem transmisji danych.



## Sterowniki – funkcje pomocnicze

**DeviceStatus DeviceStatus ()** - funkcja wykorzystywana do odczytywania statusu urządzenia, np. sprawdzanie flagi Timer'a, USART'a, itp...

może zostać wywołana przez inną funkcję sterownika lub aplikację.

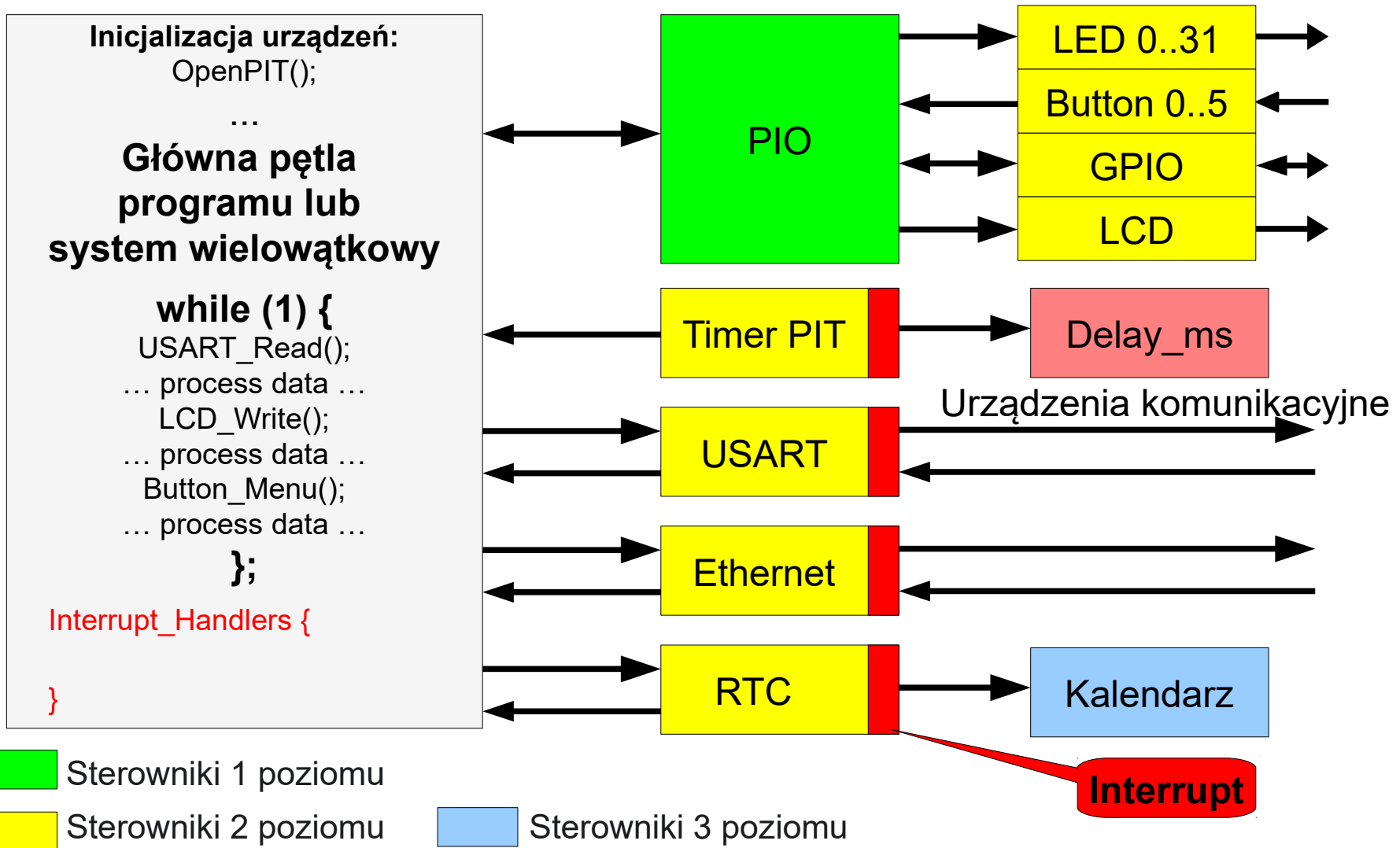
Wywołanie może nastąpić w funkcji blokującej (polling - ciągle sprawdzanie stanu urządzenia – funkcja czeka na ustawienie lub wyzerowanie flagi) lub nieblokującej (sprawdzanie stanu wywoływane w funkcji przerwania).

**Device\_INT\_Handler()** - funkcja obsługująca przerwania od urządzeń peryferyjnych, np. handler do timer'a PIT.

**Device\_WriteString ()** - funkcja wykorzystywana do zapisywania ciągu znaków do urządzenia. Funkcja korzysta z funkcji **Device\_Write()** zapisującej pojedynczy znak. Funkcja dziedziczy własności blokujące po funkcji niższego poziomu.



# Przykładow strukturę sterowników systemu mikroprocesorowego





## Sterowniki systemu mikroprocesorowego

- Sterowniki urządzeń 1 poziomu:
  - Sterownik portu równoległego PIO,
- Sterowniki 2-go poziomu (korzystają ze sterowników 1-go poziomu):
  - Sterownik diod LED,
  - Sterownik klawiatury,
  - Sterownik wyświetlacza LCD,
  - Sterownik portów GPIO,
  - Sterownik Timera PIT,
  - Sterownik interfejsu USART,
  - Sterownik interfejsu Ethernet,
  - Sterownik zegara RTC.
- Sterowniki 3-go poziomu (korzystają ze sterowników 2-go poziomu):
  - Sterownik kalendarza.





## Przykładowe funkcje sterownika portu równoległego

```
PIO_Struct* PIO_Open (unsigned int *RegistersPointer, unsigned int PortMask);  
void PIO_Close (unsigned int *RegistersPointer, unsigned int PortMask);  
unsigned int PIO_Read (PIO_Struct* PoiterToPIO);  
void PIO_Write (PIO_Struct* PoiterToPIO, unsigned int Data);  
unsigned int PIO_status (PIO_Struct* PoiterToPIO);
```

### Funkcje zwracające status operacji:

```
unsigned int PIO_Read (PIO_Struct* PoiterToPIO, unsigned int *ReadData);  
unsigned int PIO_Write (PIO_Struct* PoiterToPIO, unsigned int *DataToSend);  
unsigned int PIO_Status (PIO_Struct* PoiterToPIO, unsigned int *DeviceStatus);
```

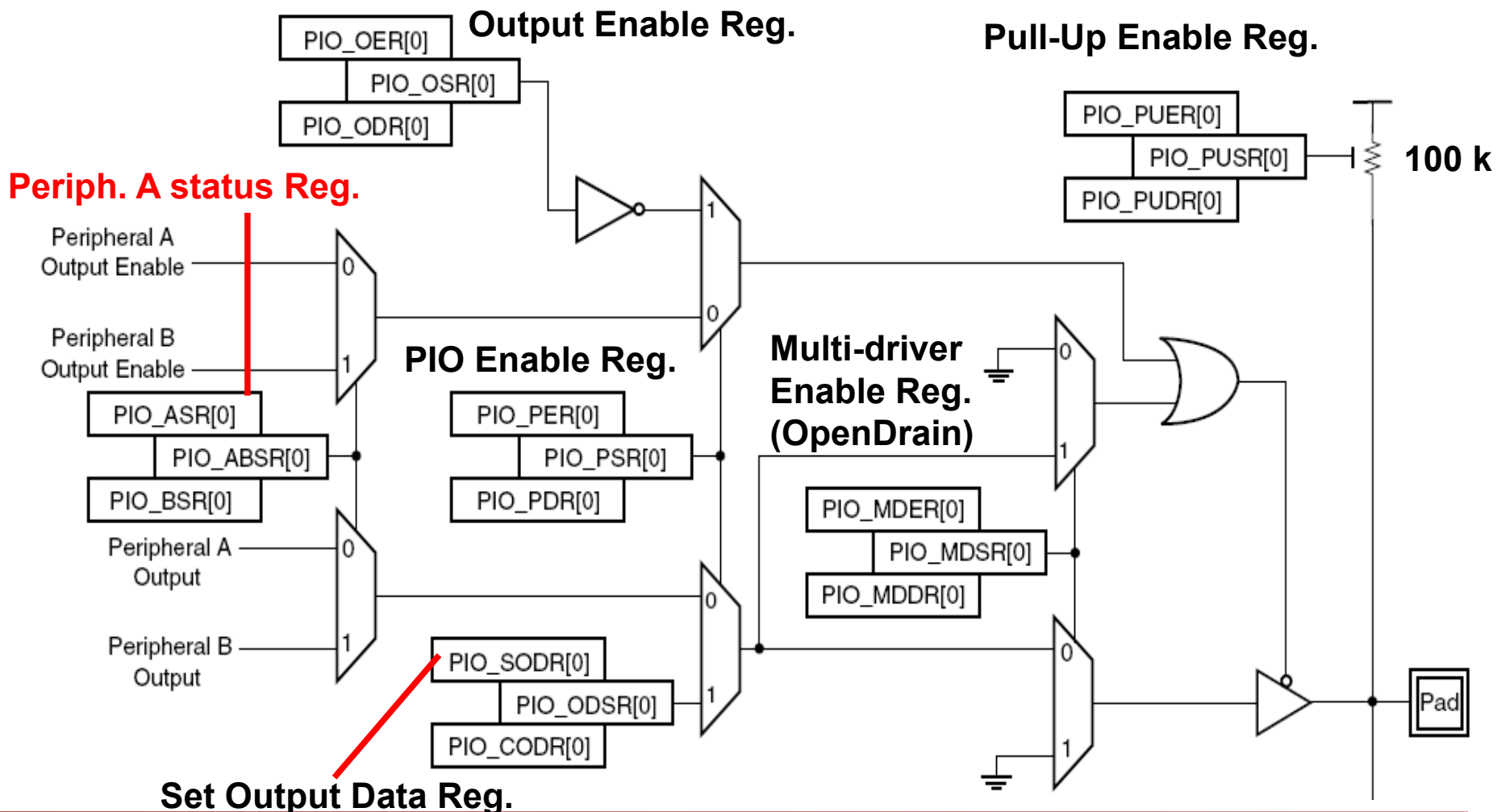
### Funkcje pomocnicze:

```
void PIO_EnablePullUp (unsigned int *RegistersPointer, unsigned int PortMask);  
void PIO_DisablePullUp (unsigned int *RegistersPointer, unsigned int PortMask);  
unsigned int PIO_StatusPullUp (unsigned int *RegistersPointer, unsigned int PortMask);
```



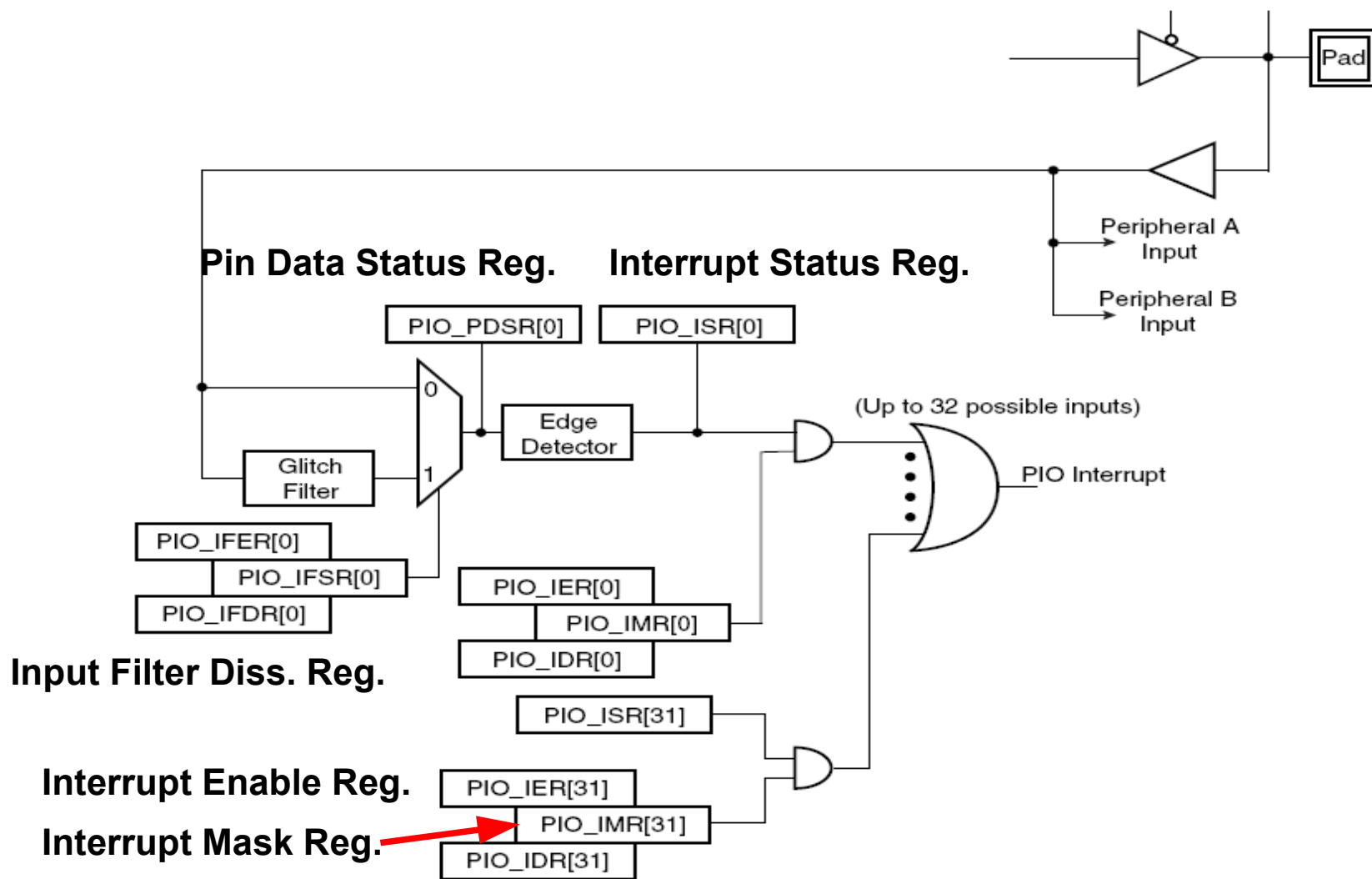
# Schemat blokowy portu I/O – sterowanie wyjściem

Figure 31-3. I/O Line Control Logic





# Schemat blokowy portu I/O – odczyt stanu wejścia





## Jak napisać sterownik ?

1. Przygotowanie struktury odzwierciedlającej rejestry danego urządzenia oraz masek pomocnych podczas operacji na rejestrach,
2. Opracowanie zmiennych pozwalających na sprawdzenie stanu danego urządzenia (np. czy urządzenie było już zainicjalizowane, czy ze sterownika korzysta jakieś urządzenie? Czy jedno?, jakie opóźnienie odmierza timer,...),
3. Opracowanie funkcji sterownika (Open, Close, Read, Write) oraz API służącego do komunikacji ze sterownikiem,
4. Opracowanie procedur obsługujących przerwania (wcześniejsze funkcje powinny na tym etapie działać – późniejsza lokalizacja problemów z włączonymi przerwaniem może być bardzo trudna lub nawet niemożliwa)

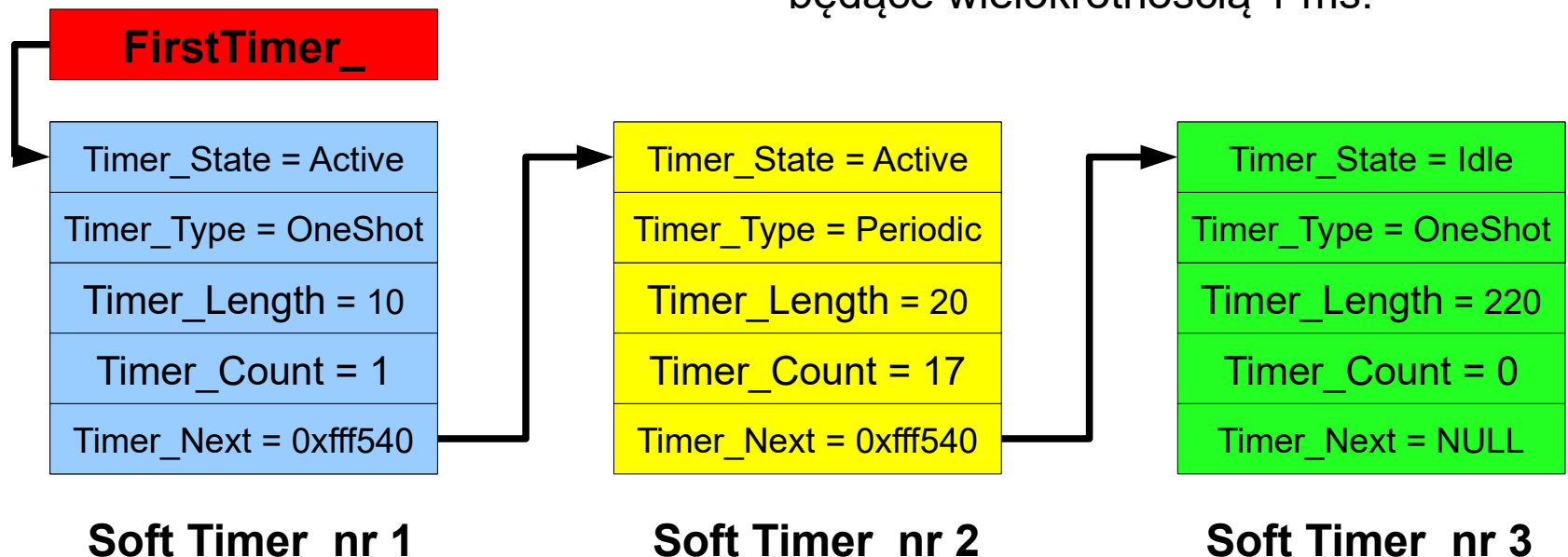


# Sterownik obsługujący kilka timer'ów

```
struct {  
    TimerState      Timer_State;           /* current timer state */  
    TimerType       Timer_Type;           /* current timer mode */  
    unsigned int    Timer_Length;        /* length of delay - number of hardware timer ticks */  
    unsigned int    Timer_Count;        /* number of ticks to expire for each software timer */  
    Timer *         Timer_next;         /* pointer to the next software timer */  
} FirstTimer, *FirstTimer_;
```

## Uporządkowana lista timer'ów

**Timer sprzętowy generuje przerwanie co 1 ms.**  
Timery programowe mogą generować przerwania będące wielokrotnością 1 ms.





## Przykładowy sterownik timer'a

```
enum TimerState {Idle, Active, Done};
enum TimerType {OneShot, Periodic};
typedef struct {
    TimerState    Timer_State;    /* current timer state */
    TimerType     Timer_Type;     /* current timer mode */
    unsigned int  Timer_Length;   /* length of delay - number of hardware timer ticks */
    unsigned int  Timer_Count;    /* number of ticks to expire for each software timer */
    Timer *       Timer_next;    /* pointer to the next software timer */
} Timer, *Timer_;

int Timer_Open(Timer_ * TPoin)    /* configure hardware and soft timer */
int Timer_Close(Timer_ * TPoin)   /* release hardware or soft timer */
int Timer_Start(unsigned int milliseconds, TimerType Type, Timer_ * TPoin) /* start timer */
int Timer_Wait_For (Timer_ * TPoin)/* wait until timer fired */
void Timer_Cancel (Timer_ * TPoin) /* turn off software timer */

static void Timer_INT (void);     /* hardware timer interrupt, e.g. 1 ms */
```



## Funkcje sterownika Timer'a programowego (1)

```
int Timer_Start (unsigned int milliseconds, TimerType Type, Timer_ * TPoin){
    if (Tpoin->Timer_State != Idle)
        return -1;
    Tpoin->Timer_State = Active;
    Tpoin->Timer_Type = Type;
    Tpoin->Timer_Length = milliseconds / MSPERTICK;    /* delay in ms */
    AddTimerToList (Tpoin);                          /* add pointer to the previous timer structure */
    return 0;
}

void Timer_Cancel (Timer_ * TPoin){
    if (Tpoin->Timer_State == Active)
        RemoveTimerFromList (Tpoin);
    Tpoin->Timer_State = Idle;
}
```



## Funkcje sterownika Timer'a programowego (2)

```
int Timer_Wait_For (Timer_ * TPoin){
    if (Tpoin->Timer_State != Active)
        return -1;
    while (Tpoin->Timer_State != Done);
    if (Tpoin->Timer_Type = Periodic){
        Tpoin->Timer_State = Active;
        AddTimerToList (Tpoin);
    }
    else
    {
        Tpoin->Timer_State = Idle;
    }
    return 0;
}
```





## Obsługa przerwania od timera sprzętowego

```
static void Timer_INT (void) { /* hardware timer interrupt, e.g. 1 ms */
```

0. Obsługa timera sprzętowego (reinicjalizacja  $t = 1$  ms, potwierdzenie przerwania, itd...)

1. Sprawdź listę timerów,
2. Zdekrementuj pola `Timer_Count`,
3. Jeżeli `Timer_Count` równe 0 i `TimerType = OneShot` usuń timer z listy,
4. Jeżeli `Timer_Count` równe 0 i `TimerType = Periodic` uruchom timer ponownie, `Timer_Count = Timer_Length`.
5. Modyfikacja flagi od danego timer'a (`Timer_Fired`) lub wygenerowanie przerwania programowego od danego timera.

```
}
```



## Sterowniki, a język C++

```
enum TimerState { Idle, Active, Done };
enum TimerType { OneShot, Periodic };
class Timer {
public:
    Timer ();
    ~Timer ();

    int Start (unsigned int milliseconds, TimerType = OneShot);
    int Wait_For ();
    void Cancel ();

    TimerState      State;
    TimerType       Type;
    unsigned int    Length;

    unsigned int    Count;
    Timer *         pNext;
private:
    static void INT ();
};
```



## Przykładowe pytania (1)

- Interfejs I2C jest interfejsem:
  - Równoległym,
  - Szeregowym,
  - Umożliwiającym transmisję danych na duże odległości rzędu dziesiątek metrów,
  - Umożliwiającym transmisję danych z szybkością kilku Mbps,
  - Umożliwiającym transmisję Master – Slave,
  - Umożliwiającym transmisję Multi Master – Slave,
  - Umożliwiającym transmisję Master – Multi Slave,
  - W którym ramka danych zawiera bit startu oraz bit stopu,
  - W którym ramka danych zawiera bit parzystości,
  - Umożliwia transmisję od 5 do 9 bitów w jednej,
  - Umożliwia adresowanie urządzeń przy użyciu 8-bitowego adresu,
  - Umożliwia adresowanie urządzeń przy użyciu 10-bitowego adresu
  - Wymaga konwersji napięć odpowiadających przesyłanym symbolom MARK i SPACE określonych w standardzie,
  - Pozwala na transmisje danych typu Full-duplex, Half-duplex, itd...



## Przykładowe pytania (2)

Na rys. 1 podano schemat blokowy timera. Okres czasu odmierzanego przez timer wyraża się wzorem: ..... . Proszę policzyć:

- Okres maksymalnego opóźnienia generowanego przez timer,
- Jaką wartość należy wpisać do rejestru DIV, aby timer odliczył czas równych 1 ms?
- Jaka jest wartość błędu popełnionego podczas odmierzenia odcinka czasu równego 8 ms?

Na rys. 2 przedstawiono schemat blokowy generatora częstotliwości wzorcowej zbudowanego z wykorzystaniem pętli synchronizacji fazy (PLL).

- Proszę policzyć jaką częstotliwość wygeneruje pętla dla wartości wpisanych do rejestrów równych odpowiednio DIV = 0x0A i MUV = 0x56.
- Jaką wartość należy wpisać do rejestrów, aby uzyskać częstotliwość  $f = 1$  MHz, dla częstotliwości zegara wzorcowego równej  $f = 32\,768$  Hz.

Procesor wykonuje następujący program: **while (1) {};**. Proszę omówić operacje jakie wykona procesor w po wygenerowaniu przerwania od układu Timera. Procedura obsługi timera:

```
void Timer_INT (void) {  
    TimerPointer->INT_FLAG = 0;  
}
```



# Startup File



## Structure of Startup File

**Startup file** code is executed by processor just after reset. The code is responsible for configuration of basic peripheral devices and modules of processor.

Startup file is usually written in assembler because the code requires access to all processor resources that cannot be accessed from higher level languages. The code is executed before high level code (e.g. C/C++). Startup file is responsible for:

- Allocation of memory and configuration of stack pointers for different modes of operation (user, supervisor, IRQ, FIQ, etc...),
- Configuration of memory (remap FLASH memory, activate SRAM/DRAM, clean memory),
- Initialise exception vectors,
- Copy of Operating System or application code to memory,
- Initialise global variables in RAM (copy data from ROM, init variables with 0s),
- Configure peripheral devices,
- Initialise interrupt controller,
- Change processor mode if required,
- Call int main (void) function.

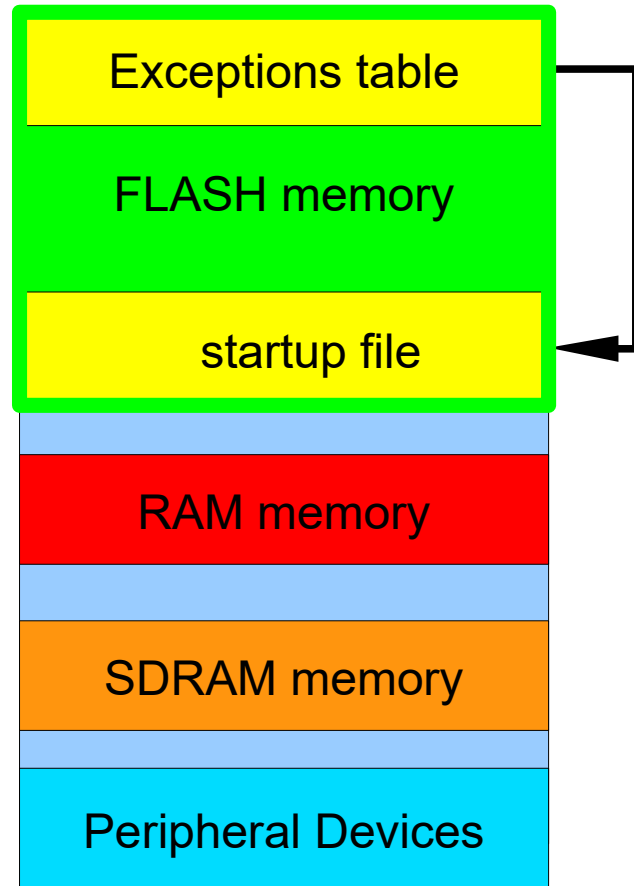


# Structure of Startup File (2)

```
RESET vector (address 0)    0x0000.0000
.section .text
reset_handler:
ldr  pc, =_low_level_init
/*  Initialization...  */
_low_level_init:

_stack_init:                0x0030.0000
_init_data:
_init_bss:                  0x2000.0000

_branch_main:              0xFFFF.F000
```





## Structure of Startup File (3)

Program in the main loop cannot be finished. Processor cannot execute return or exit functions.

...

...

**\_branch\_main:**

```
ldr    r0, =main
mov    lr, pc
bx     r0
```

...

...

```
void main (int) {
    While (1)
    {
```

main program

```
}
```

```
return 0;
```

```
}
```





## Structure of Startup File (4)

### Configuration of essential devices, required for processor operation:

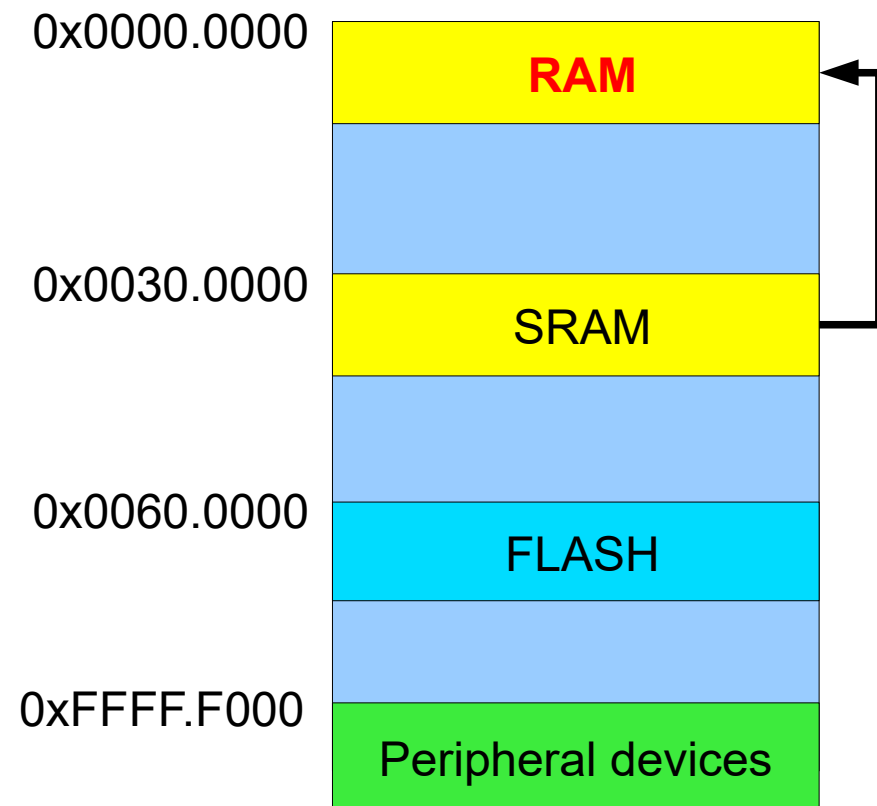
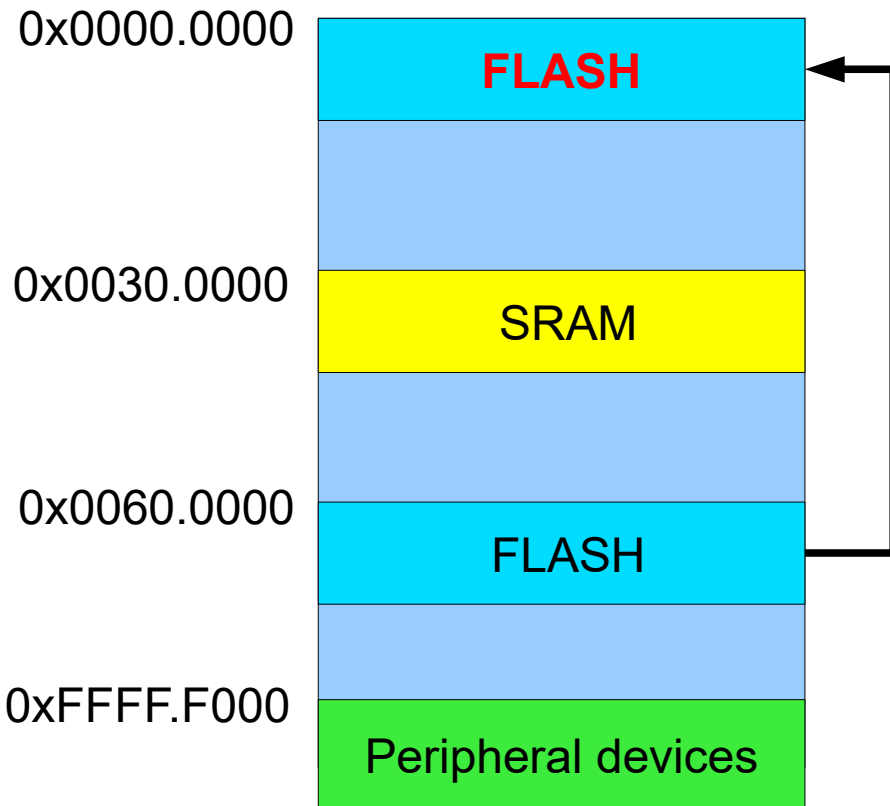
- Configuration of reference clock module (PLL). After reset processor operates with “slow clock” (internal RC generator),
- Configuration of memory controller (FLASH, RAM) – configure number of WaitStates, base address,
- Remap memory FLASH<->SRAM,
- Configure Watch-Dog timer (after reset Watch-Dog is active),
- Configure AIC module (assign default interrupt handlers),
- Initialise stack pointers for different modes of operation (user, IRQ, FIQ,...),
- Unblock NRST input (external reset).



# Remapping of Memory

Map of memory during reset

Map of memory after remap



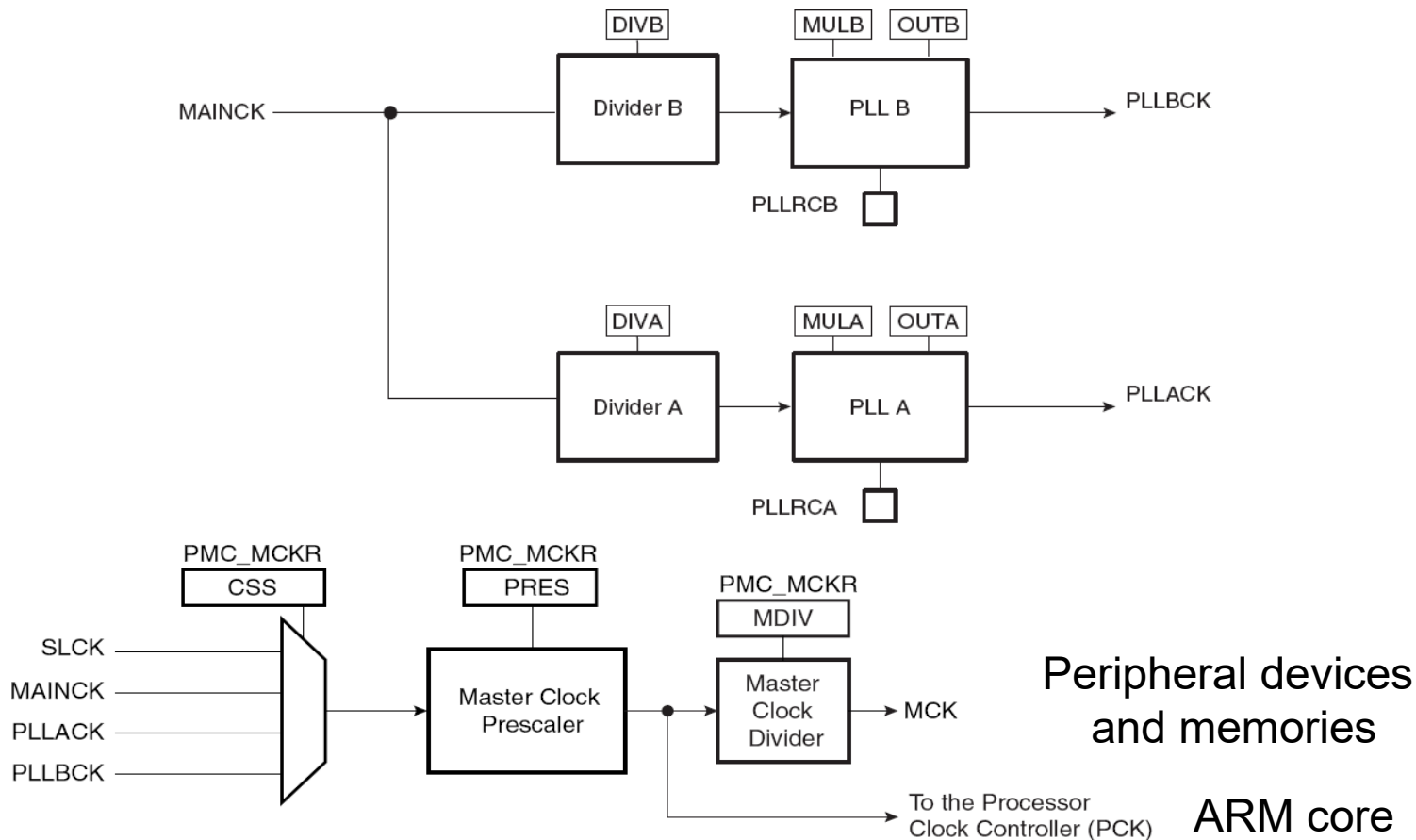
Remapping of FLASH memory is performed after execution of startup code (REMAP register)



# Configuration of Reference Clock – Chapter 28 (1)

After reset processor operates with slow clock with frequency  $f = 32768$  Hz. Slow clock is always active (generated by build-in RC generator).

After reset Phase Locked Loops (PLLs) are disabled.





## Generator with PLL (1)

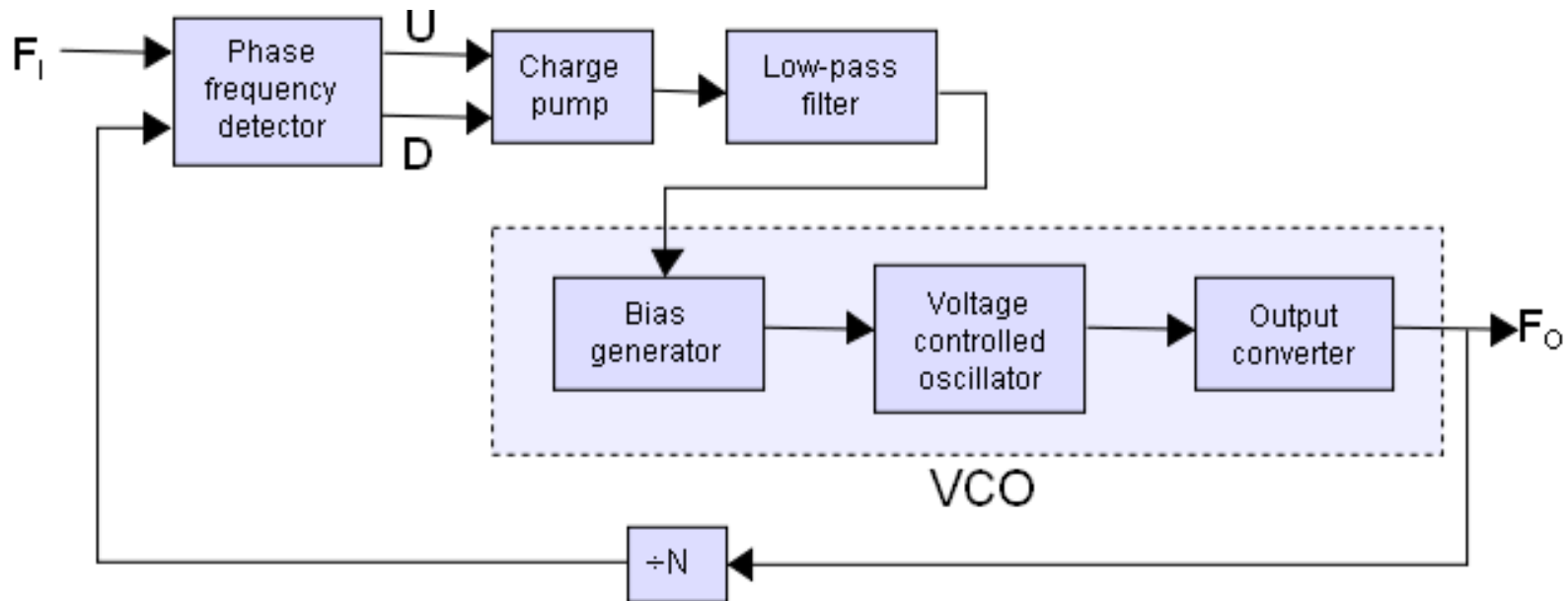
**Phase Locked Loop (PLL)** – electronic circuit based on feedback used for automatic regulation of generator frequency. PLL is used in frequency synthesizers, heterodyne RF receivers, frequency sources and frequency multipliers.

### PLL generator is composed of:

- Reference generator (quartz generator),
- Phase detector,
- Low pass filter,
- Voltage Controlled Oscillator (VCO),
- Feedback loop with suitable frequency divider.



## Generator with PLL (2)



High frequency signal generated by VCO is connected to output ( $F_o$ ). The same signal is connected to divider ( $/N$ ). The output of divider is connected to phase detector (PD) and compared with stable reference signal. Phase difference of reference signal and output signal divided by  $N$  is used to control VCO. Feedback loop strive to obtain synchronous signals (phase error equal to 0). Low pass filter is required to make the loop stable.



## Configuration of Reference Clock (2)

### Procedure to turn on PLL generator:

1. Turn on quartz generator. Wait until frequency will be stable (bit PMC\_MOSCS).
2. Configure PLLA,  $f = 16\,367\,660 \cdot 110 / 9 = \sim 200$  MHz. After turning on PLLA wait until PLLA will be stable (bit PMC\_LOCKA) and frequency will be also stable (bit PMC\_MCKRDY).
3. Switch processor to use PLLA (in example additional divider by 2), bit AT91C\_PMC\_CSS\_PLLA\_CLK. Wait until frequency will be stable.

### Konfiguracja PLLA:

AT91C\_BASE\_PMC->

```
    PMC_PLLAR = AT91C_CKGR_SRCRCA | /* programming PLL */
    AT91C_CKGR_OUTA_2 | /* electrical parameters */
    (0x3F << 8) | /* counter = 63 */
    (AT91C_CKGR_MULA & (0x6D << 16)) | /* multiplier 109 */
    (AT91C_CKGR_DIVA & 9); /* divider 9 */
```

$f_{\text{ref}} = 16\,367\,660$  Hz

$f_{\text{out}} = f_{\text{ref}} \cdot (MULA+1) / DIVA = 16\text{ MHz} \cdot 110 / 9 \Rightarrow 200\text{ MHz}$

$f_{\text{MCK}} = f_{\text{out}} / 2 \Rightarrow \sim 100\text{ MHz}$

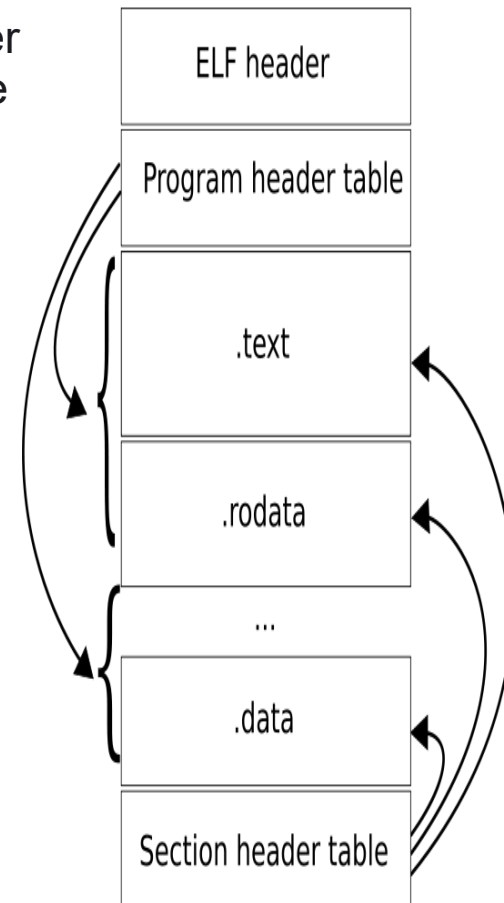


# Analysis of lowlevel.c file



## COFF vs ELF

- **COFF (Common Object File Format)** – specification of format for executable, object code, and shared library computer files used on Unix systems. COFF was introduced to substitute old **a.out format**. COFF is used also on other platforms, e.g. Windows. Currently, ELF standard is promoted instead of COFF.
- **ELF (Executable and Linkable Format)** – common standard file format for executables, object code, shared libraries, and core dumps used on different architectures, e.g.: x86 family, PowerPC, OpenVMS, BeOS, PlayStation Portable, PlayStation 2, PlayStation 3, Wii, Nintendo DS, GP2X, AmigaOS 4 and Symbian OS v9.
- Usufull tools:
  - readelf
  - elfdump
  - objdump



Źródło: wikipedia





# Linker Script

```
/* elf32-littlearm.Ids for ARM At91SAM9263 */
OUTPUT_FORMAT ("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH (arm)
ENTRY (reset_handler)
/*#include "project.h"*/
SECTIONS
{
    . = 0x1.0000;           /* base address */
    .text : { (.text) }    /* code section */
    . = 0x800.0000;        /* base address */
    .data : { (.data) }    /* initialized data */
    .bss : { *(.bss) }     /* uninitialized data */
}

LED_test: $(OBJJS)

$(LD) $(LDFLAGS) -Ttext 0x20000000 -Tdata 0x300000 -n -o $(OUTFILE_SDRAM).elf $
(OBJJS)
```



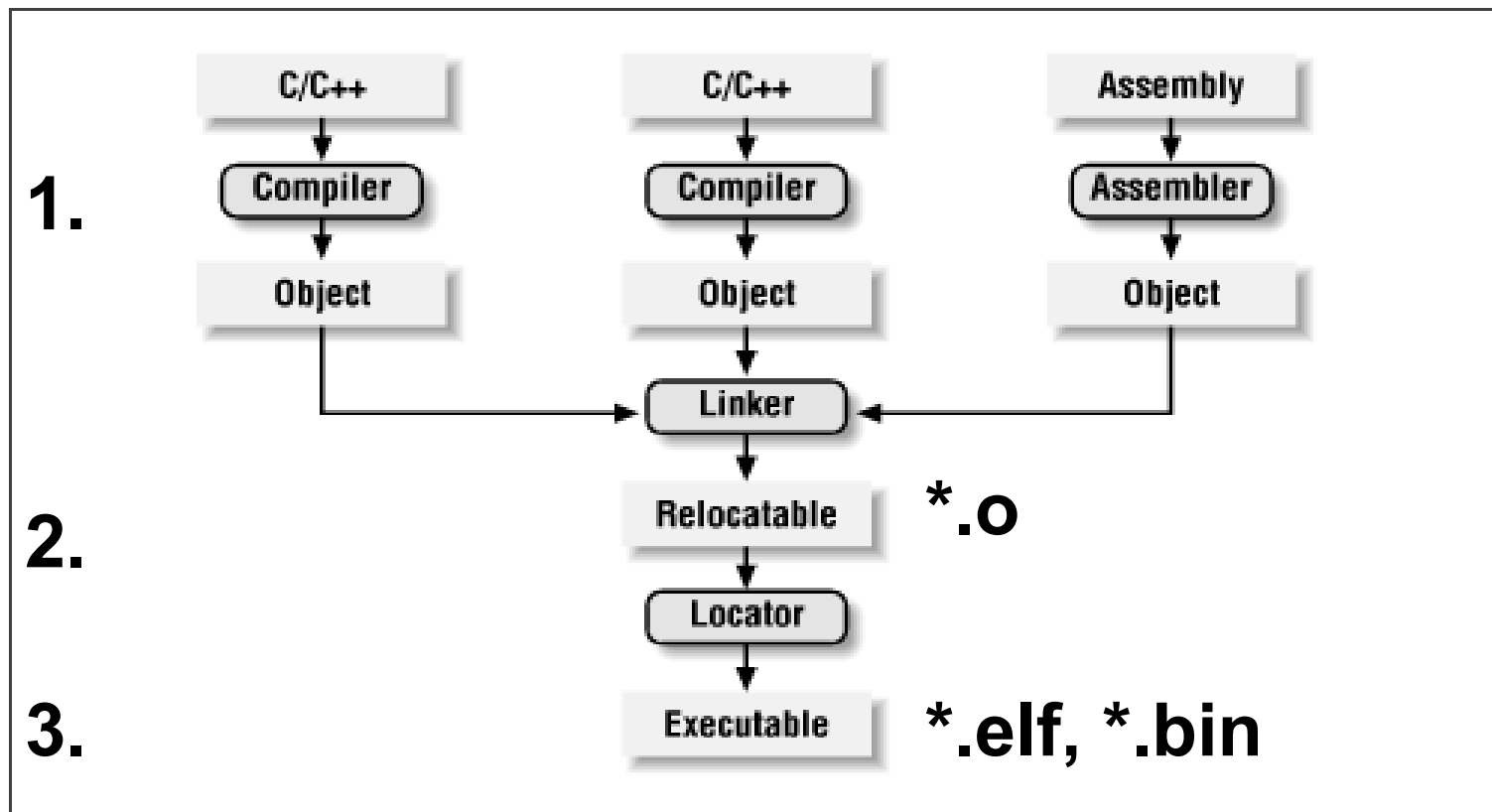
# Linker script for ARM AT91SAM9263 processor

```
SECTIONS {
  .text : {
    _stext = .;
    *(.text)      /* program code */
    *(.rodata)    /* read-only data (constants) */
    *(.rodata*)
    . = ALIGN(4);
    _etext = .; }
  /* all initialized .data that go into FLASH */
  .data : AT ( ADDR (.text) + SIZEOF (.text) ) {
    _sdata = .;
    *(.vectors) /* vectors table */
    (.data)     /* initialized data */
    _edata = .; }
  /* all uninitialized .bss that go into FLASH */
  .bss (NOLOAD) : {
    . = ALIGN(4);
    _sbss = .;
    *(.bss)          /* uninitialized data */
    _ebss = .; } }
end = .;
```

```
CROSS_COMPILE=arm-elf-
LD=$(CROSS_COMPILE)gcc
LDFLAGS+=-nostartfiles -WI,--cref
LDFLAGS+=-lc -lgcc
LDFLAGS+=-T elf32-littlearm.lds
OBJS = cstartup.o
OBJS+= lowlevel.o main.o

LED_test: $(OBJS)
$(LD) $(LDFLAGS) -Ttext 0x20000000
-Tdata 0x300000 -n -o
$(OUTFILE_LED_test).elf $(OBJS)
```

# Compilation Process



- 1 phase – compilation of source files → relocable binary files
- 2 phase – linking of relocable files → relocable binary file
- 3 phase – generation of executable file (with assigned addresses)