Dariusz Makowski

Katedra Mikroelektroniki i Technik Informatycznych

tel. 631 2720

dmakow@dmcs.pl

http://neo.dmcs.p.lodz.pl/sw

Zakres przedmiotu

- Systemy mikroprocesorowe, systemy wbudowane
- Rodzina procesorów ARM
- Interfejsy w systemach wbudowanych
- Asembler
- Urządzenia peryferyjne
- 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 zwykłe 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.

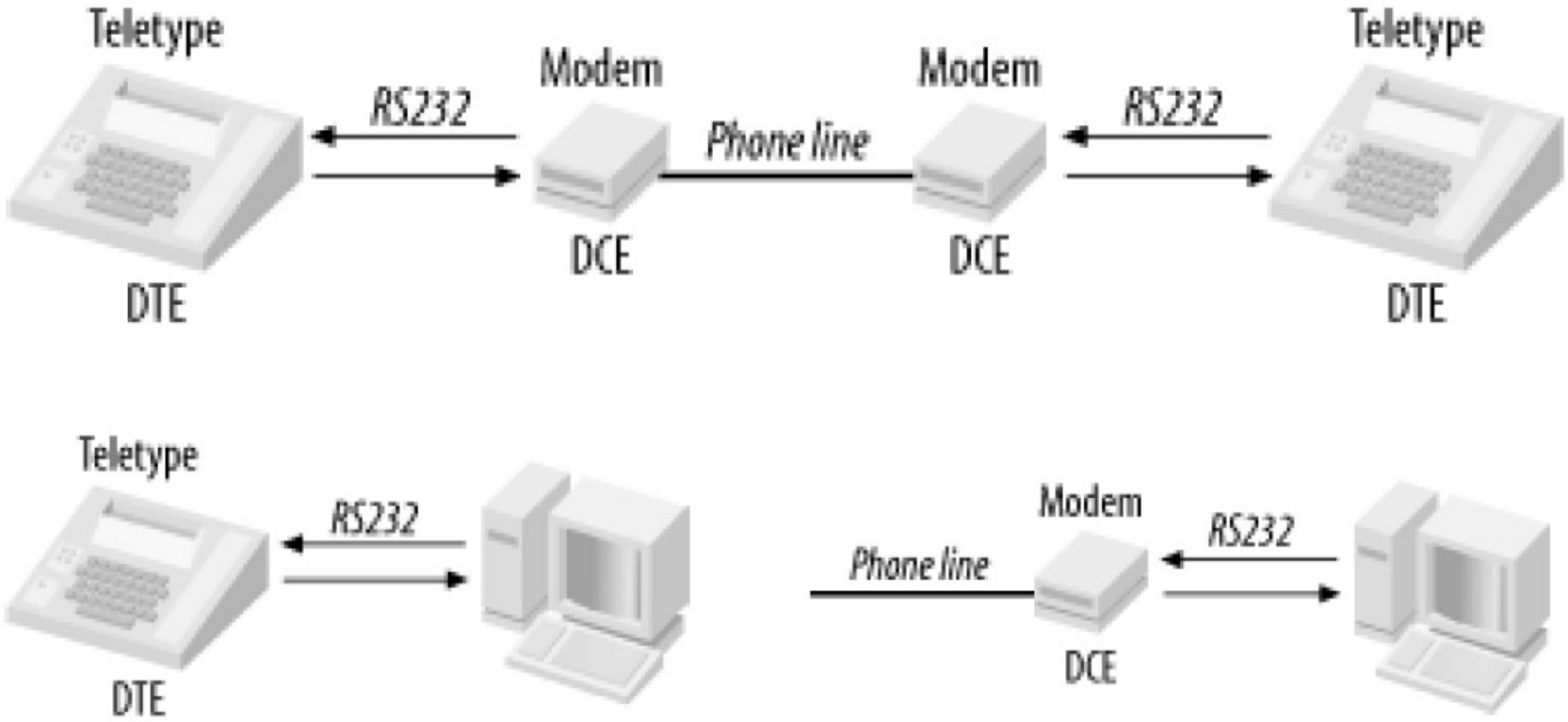
Współpraca procesora z urządzeniami peryferyjnymi

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 (1-100 Gb),
  - Magistrala PCI,
  - Magistrala PCIe (PCI express),
  - RapidIO, Serial RapidIO,

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

Interfejs szeregowy EIA RS232



Transceiver UART

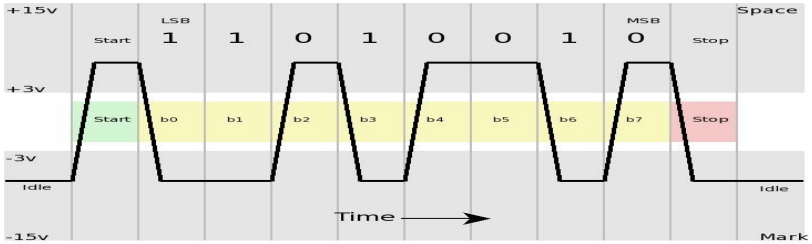
Rejestr przesuwny

D0-D7

Nadajnik

TxD

Clk



D0-D7

Odbiornik

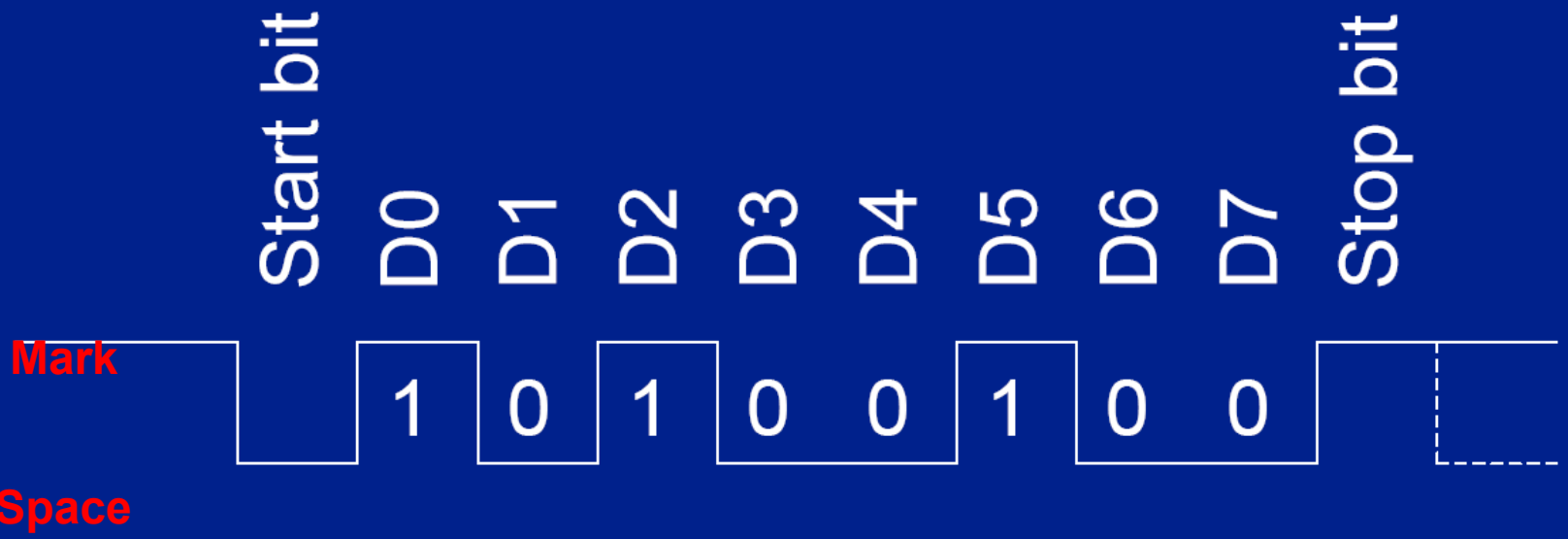
RxD

Clk

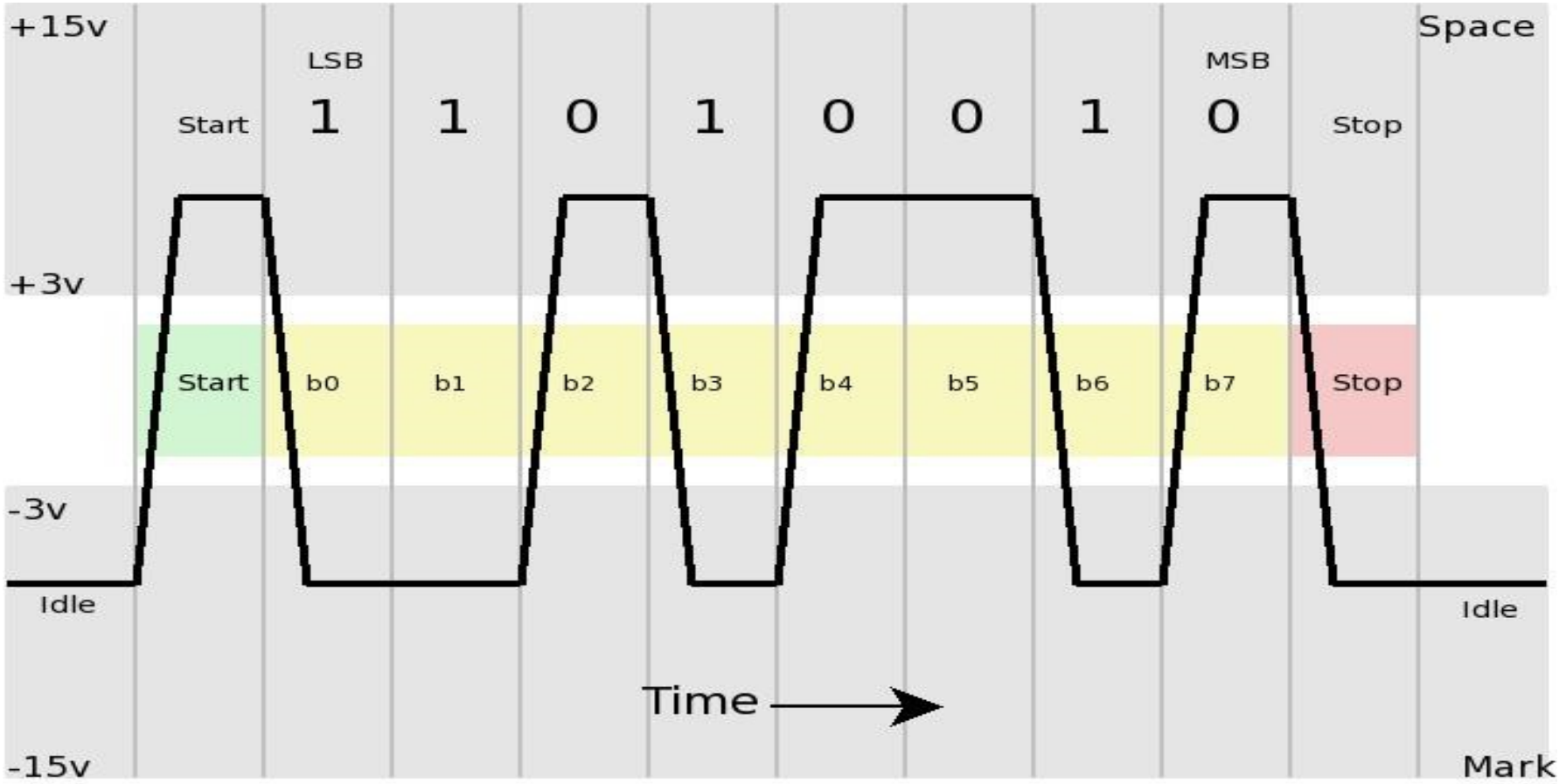
Ramka danych transmitera UART (1)

Asynchronous 8 bit waveform example

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

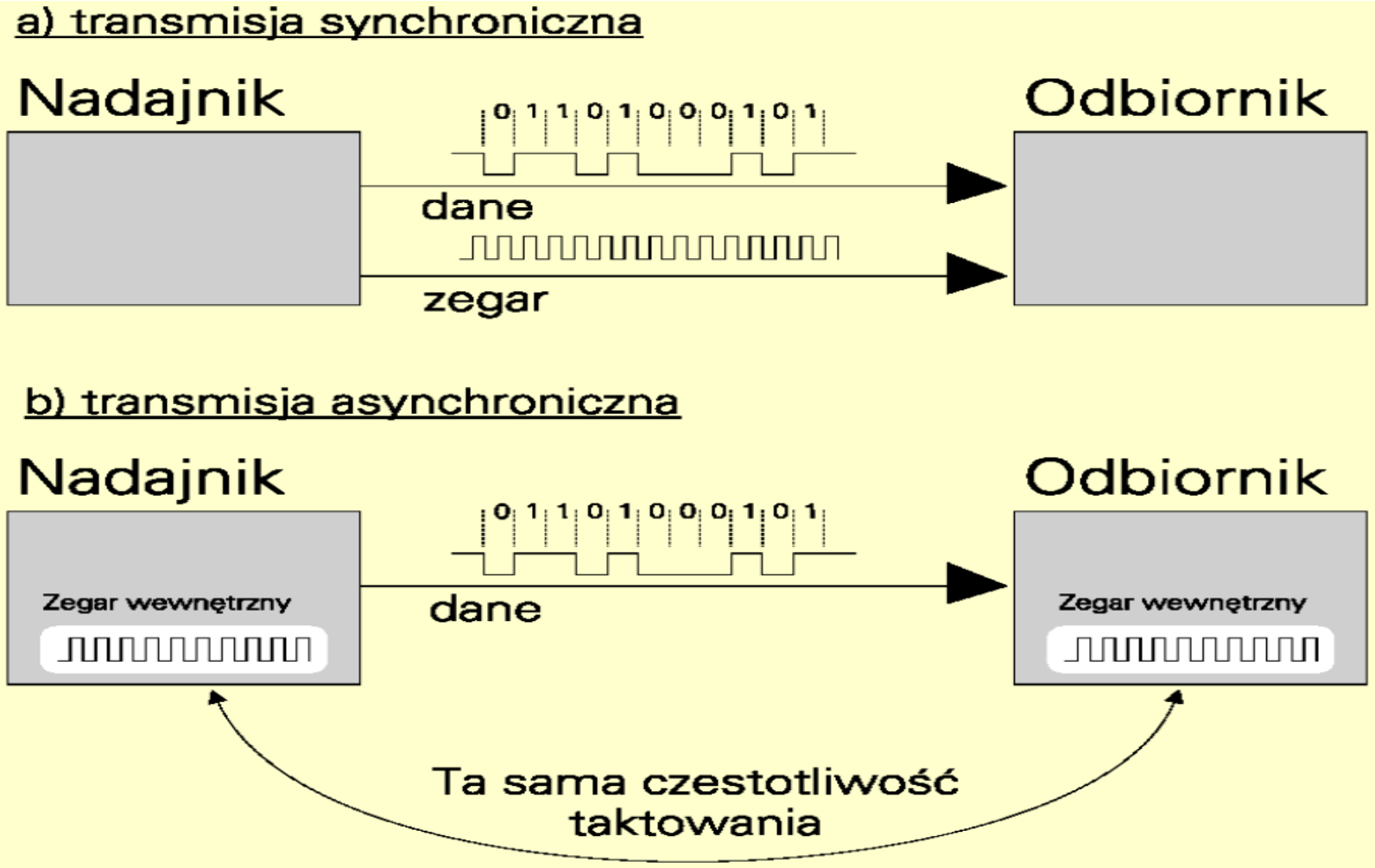


Ramka danych transmitera UART (2)



Przesylana dana: 0100.1011b = 0x4B

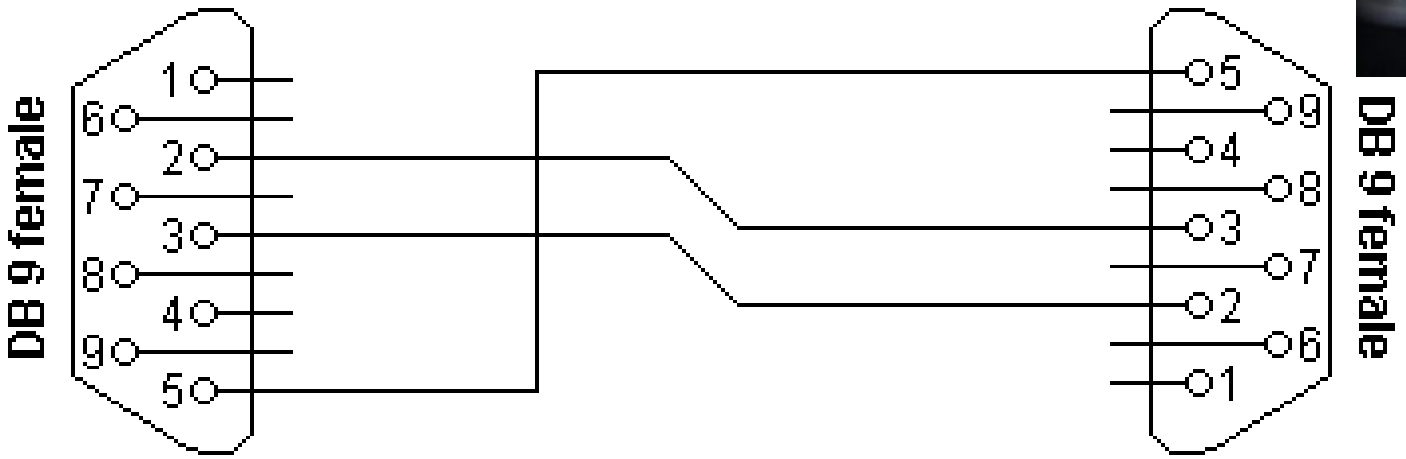
Transmisja synchroniczna vs asynchroniczna?



Specyfikacja elektryczna EIA RS232c

SPECIFICATIONS		RS232
Mode of Operation		SINGLE -ENDED
Total Number of Drivers and Receivers on One Line		1 DRIVER 1 RECVR
Maximum Cable Length		50 FT.
Maximum Data Rate		20kb/s
Maximum Driver Output Voltage		+/-25V
Driver Output Signal Level (Loaded Min.)	Loaded	+/-5V to +/-15V
Driver Output Signal Level (Unloaded Max)	Unloaded	+/-25V
Driver Load Impedance (Ohms)		3k to 7k
Max. Driver Current in High Z State	Power On	N/A
Max. Driver Current in High Z State	Power Off	+/-6mA @ +/-2v
Slew Rate (Max.)		30V/uS
Receiver Input Voltage Range		+/-15V
Receiver Input Sensitivity		+/-3V
Receiver Input Resistance (Ohms)		3k to 7k

Kabel null-modem EIA 232



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

Dodatkowe linie sterujące

Hardware Flow Control

symbol	obwód	stan linii	uwagi
DTR	108/2		komputer gotów
DSR	107		modem gotów
RTS	105		żądanie nadawania
CTS	106		gotowość do nadawania
TxD	103		transmisja danych do modemu

DTE (ang. Data Terminal Equipment) - urządzenie do przetwarzania danych (końcowe, np. komputer)

DCE (ang. Data Circuit-terminating Equipment) – urządzenie do trans. danych (np. Modem)

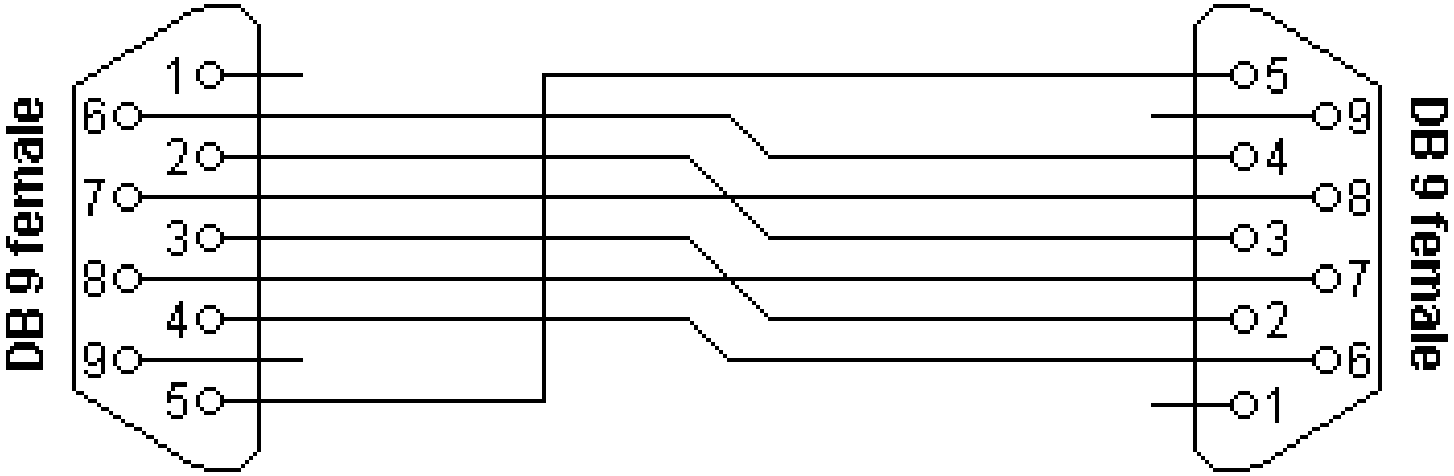
DSR - Data Set Ready - gotowość modemu

DTR - Data Terminal Ready - gotowość terminala

RTS - Request to Send Data - żądanie wysyłania

CTS - Clear to Send - gotowość wysyłania

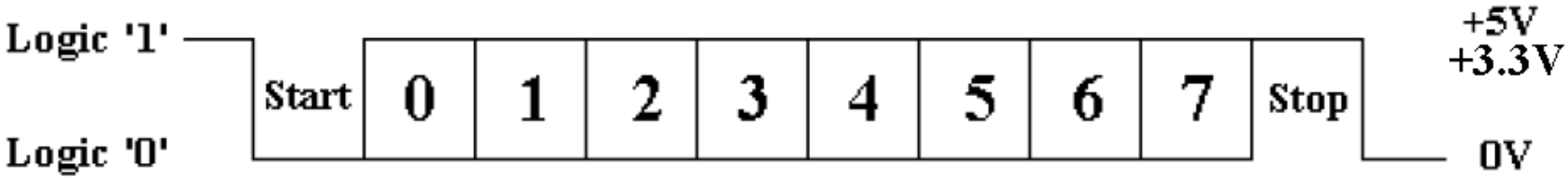
Pełny kabel null-modem



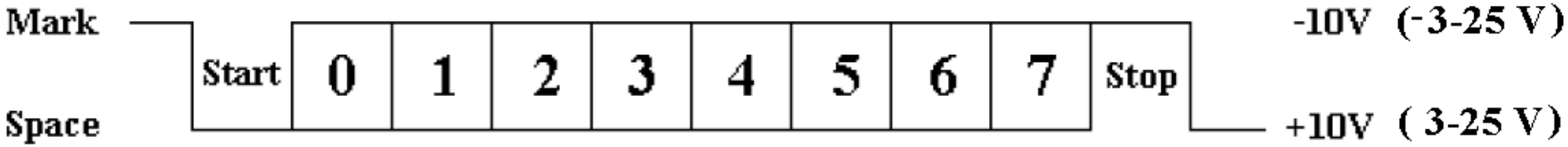
Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
4	6	DTR → DSR
5	5	Signal ground
6	4	DSR ← DTR
7	8	RTS → CTS
8	7	CTS ← RTS

Poziomy napięć określone przez standard EIA 232

Wyjście procesora

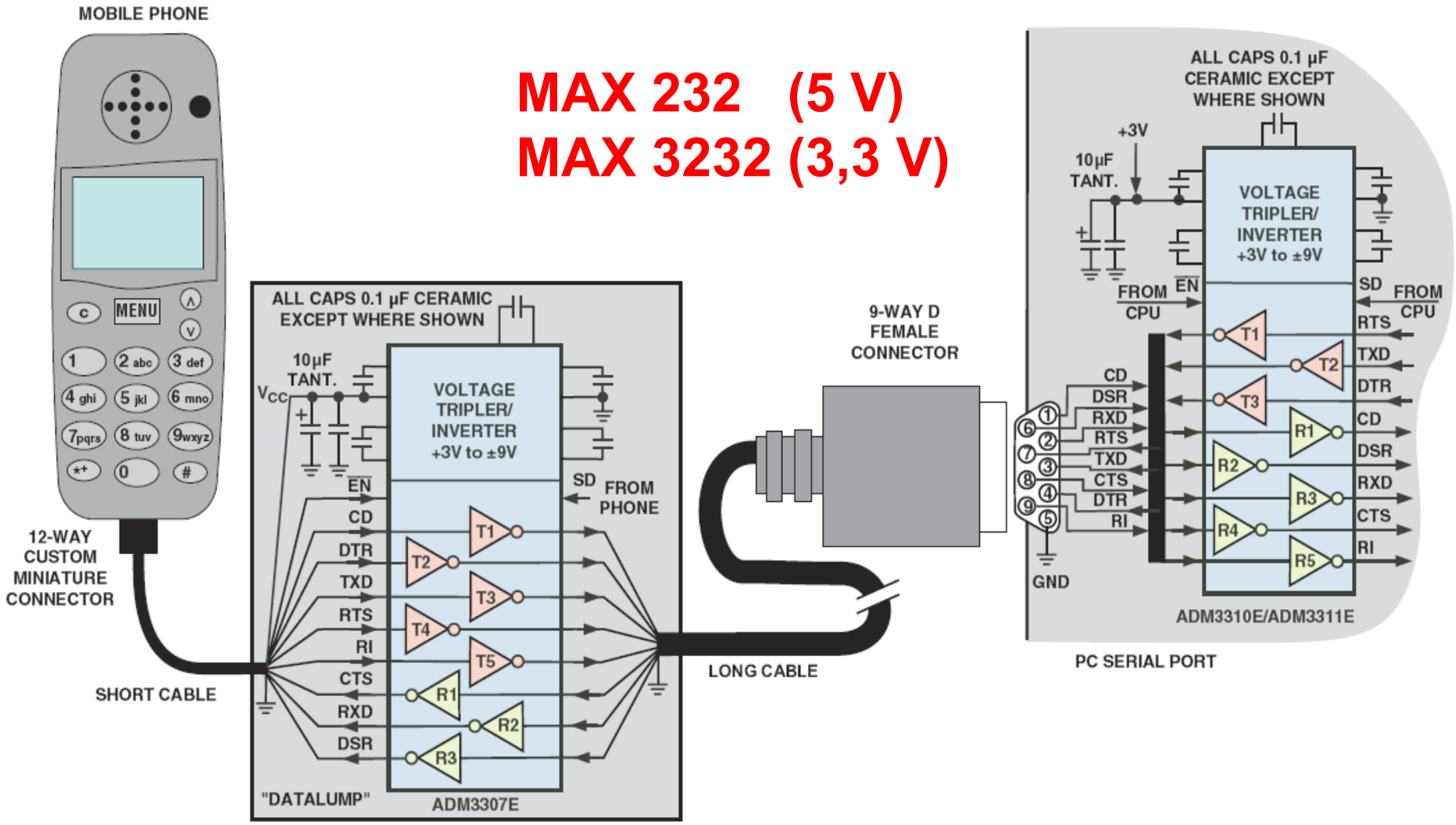


Standard EIA RS 232



RS-232 Logic Waveform

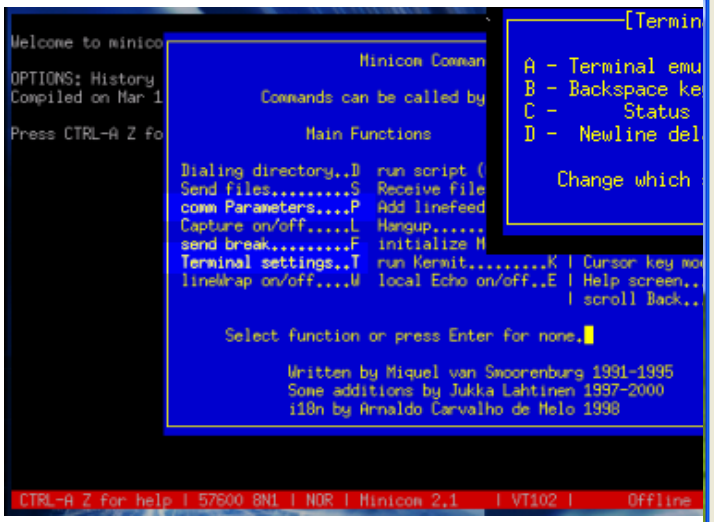
Konwerter poziomów napięć



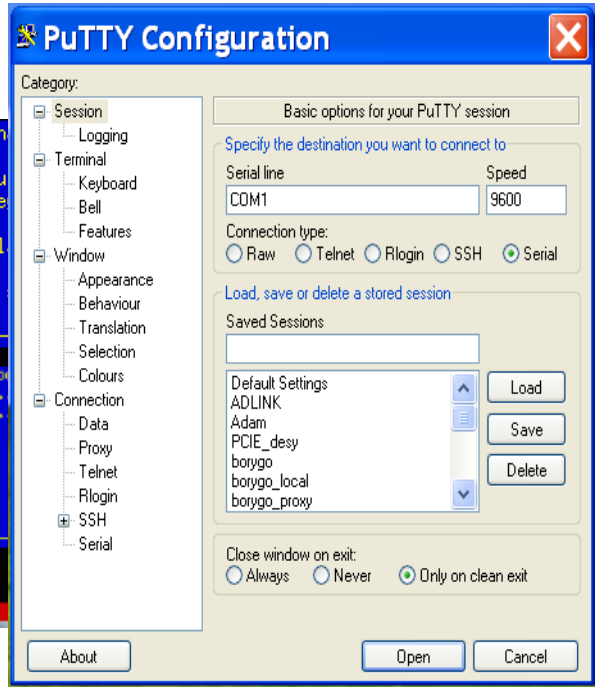
Programy do komunikacji z wykorzystaniem standardu EIA RS232

Program Hyper terminal

Program minicom



Program ssh



Program Terminal

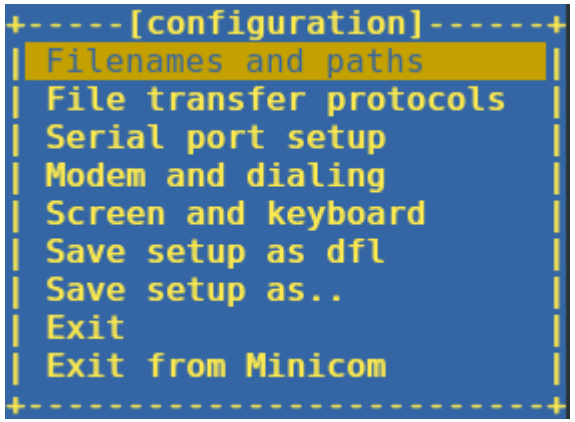
(http://www.elester-pkp.com.pl/index.php?id=92&lang=pl&zoom=0)

Program Minicom

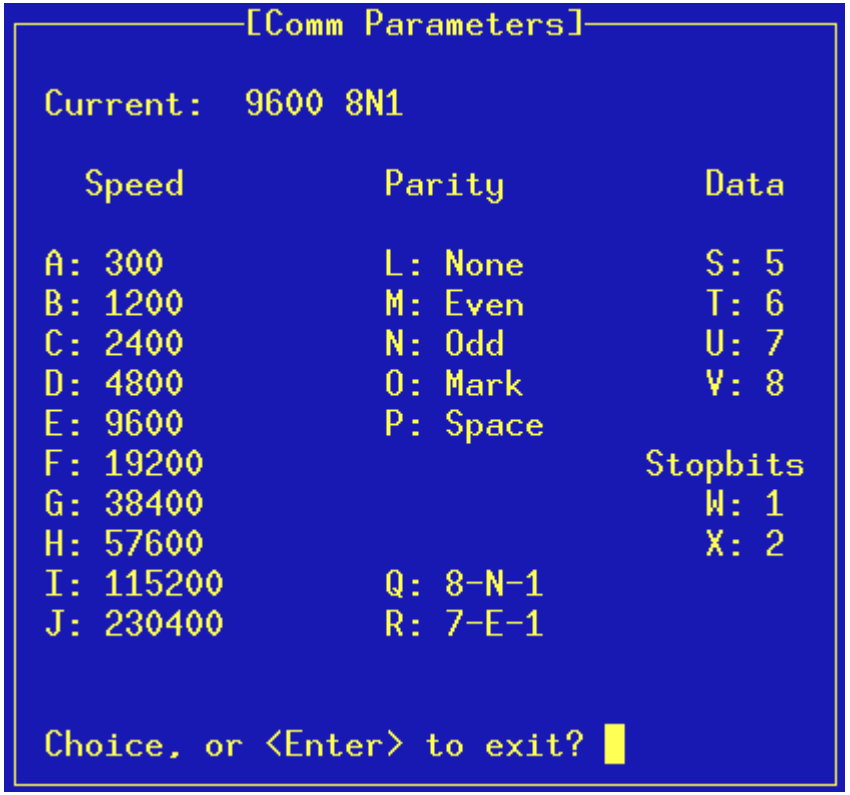
Skróty klawiszy:

- Ctrl + A → O – konfiguracja
- Ctrl + A → X – wyjście z programu
- Ctrl + A → C – wyczyszczenie ekranu

Okno konfiguracji



Okno konfiguracji parametrów transmisji



AT91SAM9263 – moduł diagnostyczny DBGU

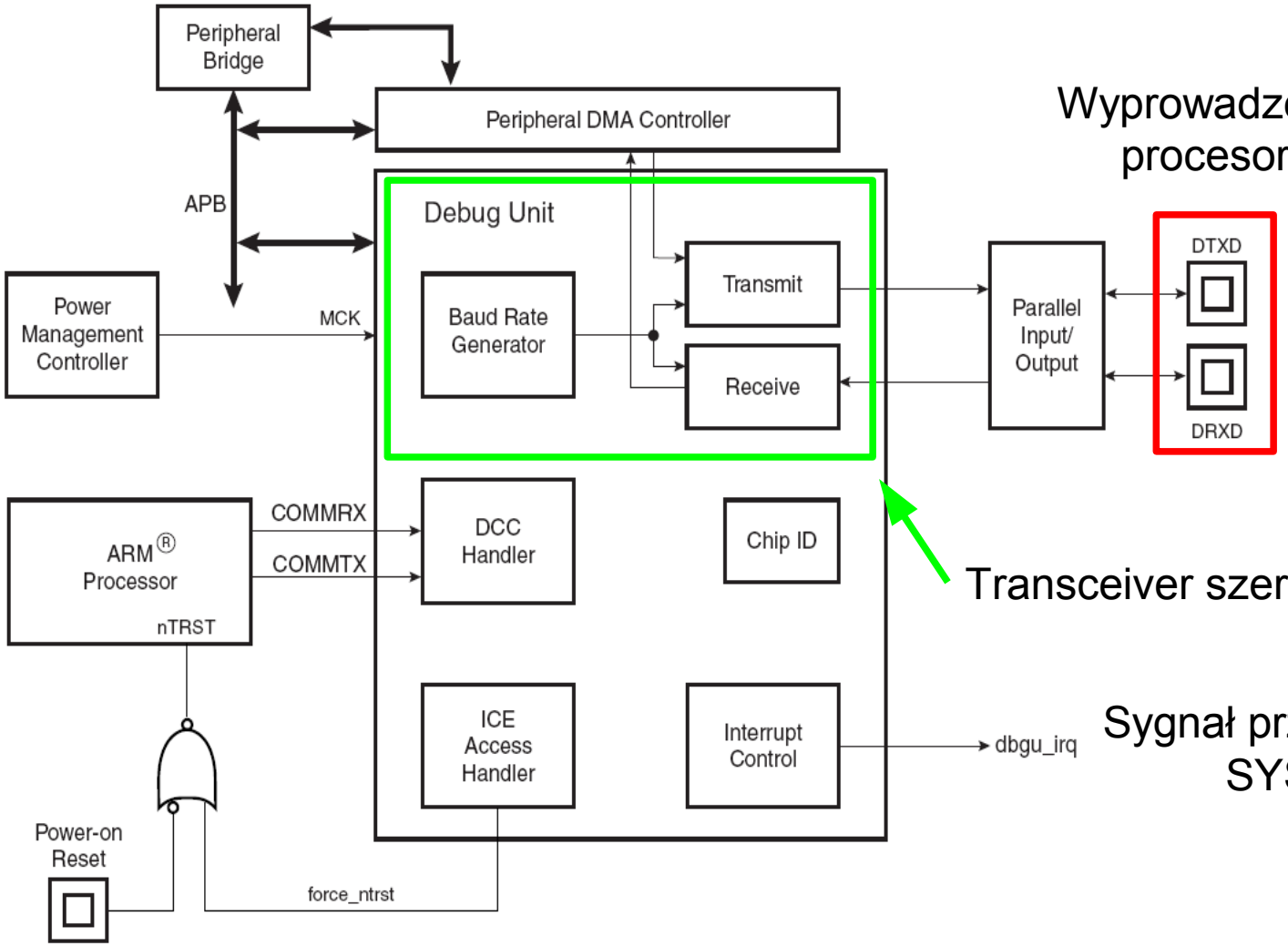
(rozdział 30)

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 przerwań systemowych współdzielonych (PIT, RTT, WDT,DMA, PMC, RSTC, MC),
- Azaliza 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

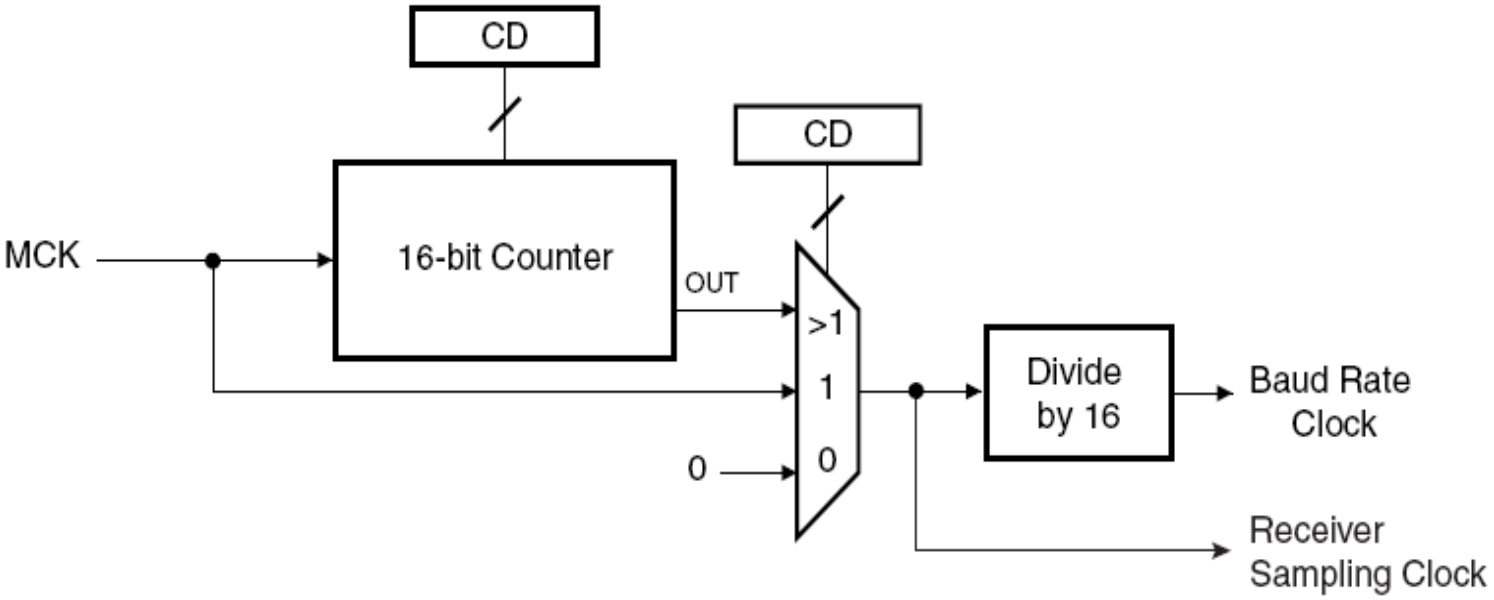


Wyprowadzenia procesora

Transceiver szeregowy

Sygnał przerwnia SYS

Szybkość transmisji



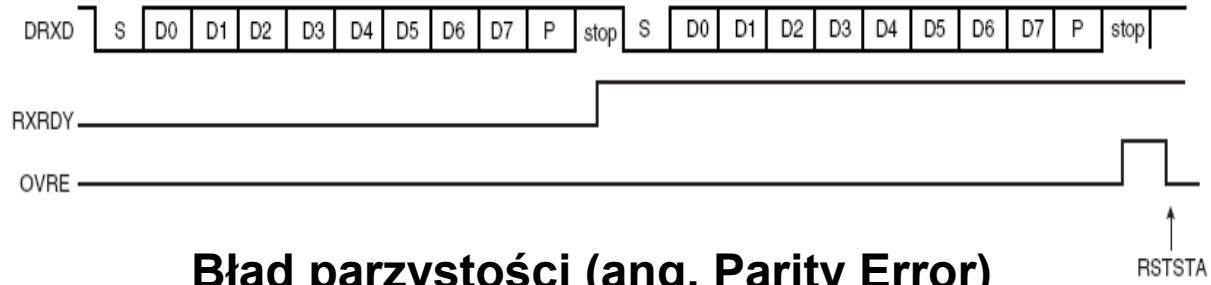
Generator sygnału zegarowego odpowiedzialnego za szybkość transmisji (ang. Baud Rate).

Szybkość transmisji danych wyrażona jest wzorem:

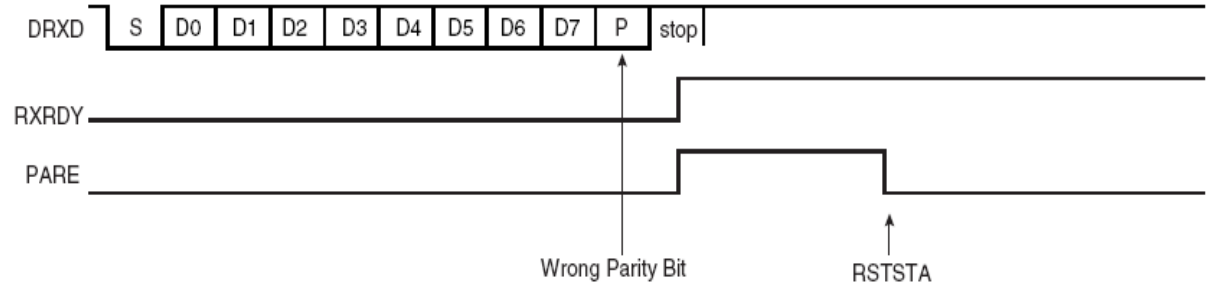
Baud Rate = MCK / (16 x CD), gdzie CD (Clock Divisor) jest polem rejestru DBGU_BRGR

Błędy podczas transmisji danych

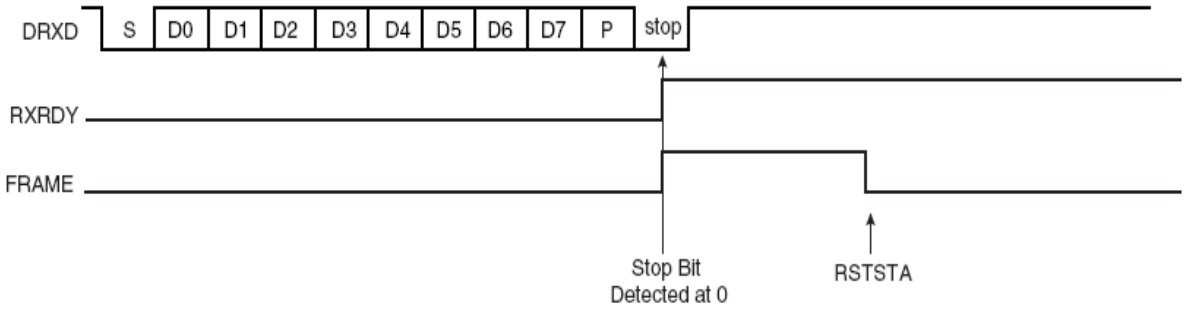
Przepełnienie bufora odbiorczego BGU_RHR (ang. Buffer Overflow)



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

void dbgu_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... )
}
```

Rejestr statusowy

30.5.6 Debug Unit Status Register
Name: DBGU_SR

Address: 0xFFFFEE14

Access Type: Read-only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

- **RXRDY: Receiver Ready**

0 = No character has been received since the last read of the DBGU_RHR or the receiver is disabled.

1 = At least one complete character has been received, transferred to DBGU_RHR and not yet read.

- **TXRDY: Transmitter Ready**

0 = A character has been written to DBGU_THR and not yet transferred to the Shift Register, or the transmitter is disabled.

1 = There is no character written to DBGU_THR not yet transferred to the Shift Register.

- **ENDRX: End of Receiver Transfer**

0 = The End of Transfer signal from the receiver Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the receiver Peripheral Data Controller channel is active.

- **ENDTX: End of Transmitter Transfer**

0 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is active.

- **OVRE: Overrun Error**

0 = No overrun error has occurred since the last RSTSTA.

1 = At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0 = No framing error has occurred since the last RSTSTA.

1 = At least one framing error has occurred since the last RSTSTA.

- **PARE: Parity Error**

0 = No parity error has occurred since the last RSTSTA.

1 = At least one parity error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0 = There are characters in DBGU_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1 = There are no characters in DBGU_THR and there are no characters being processed by the transmitter.

AT91SAM9263 – USART

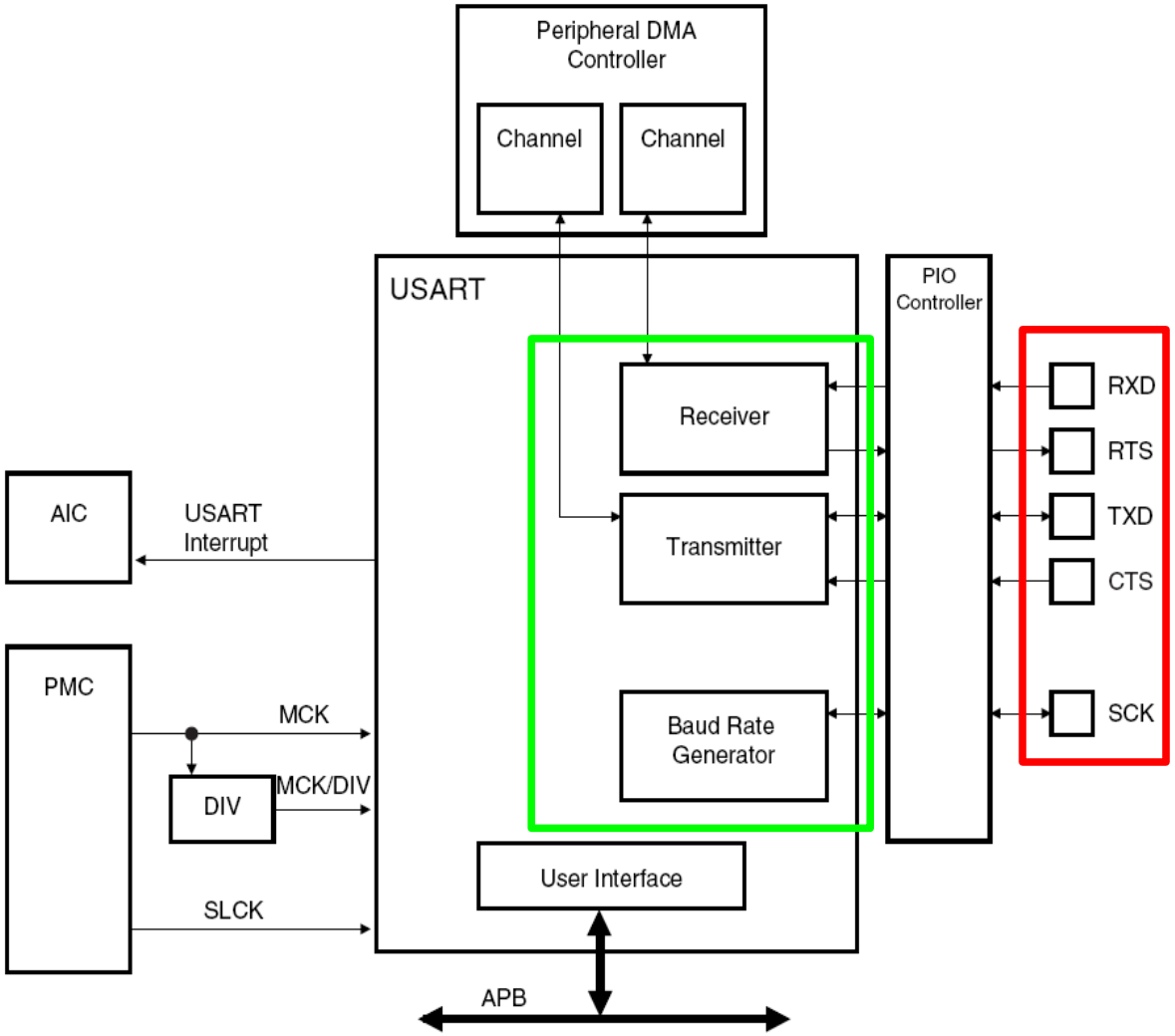
(rozdział 34)

Port szeregowy USART

Cechy portu USART (Universal Synch. Asynch. Receiver-Transmitter):

- Asynchroniczna lub transmisja danych,
- Programowalna długość ramki, kontrola parzystości, liczba bitów stopu,
- Możliwość zgłaszania przerwań systemowych współdzielonych (PIT, RTT, WDT,DMA, PMC, RSTC, MC),
- Analiza poprawności odebranych ramek,
- Sygnalizacja przepełnionego bufora TxD lub RxD,
- Możliwość odbierania ramek o zmiennej długości – wykorzystanie dodatkowego licznika do odmierzania czasu,
- Trzy tryby diagnostyczne: zdalny loopback, lokalny loopback oraz echo,
- Maksymalna szybkość transmisji rzędu 1 Mbit/s,
- Wsparcie sprzętowej kontroli przepływu danych,
- Możliwość transmisji w systemie Multidrop, transmisja danej i adresu,
- Możliwość transmisji danych z wykorzystaniem kanału DMA (Direct Memory Access),
- Wsparcie dla standardu transmisji różnicowej RS485 oraz systemów pracujących w zakresie podczerwieni (wbudowany modulator-demodulator IrDA).

Schemat blokowy transceivera USART



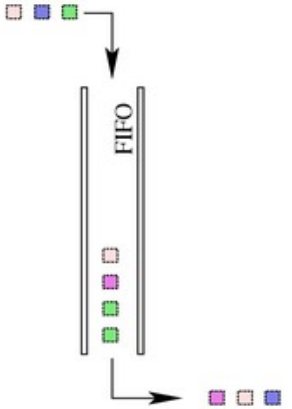
Buforowanie danych

Struktura stosu (1)

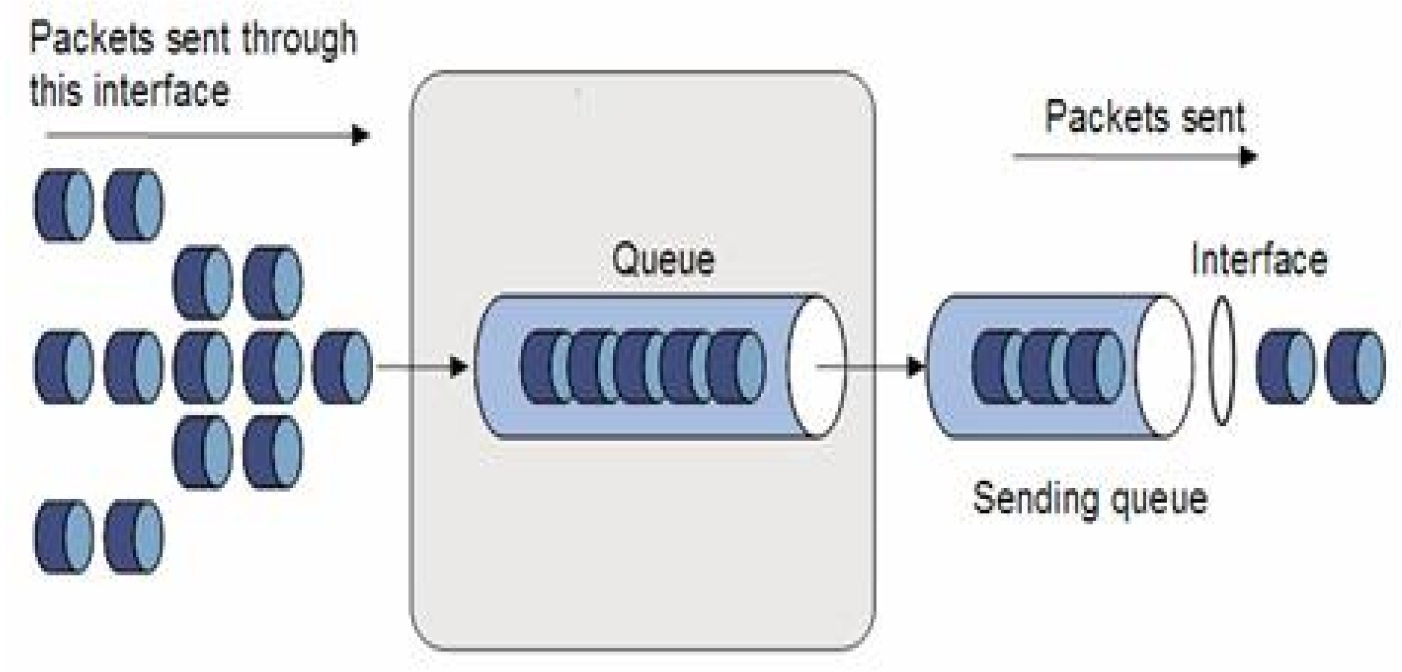
Stos (ang. stack lub LIFO Last-In, First-Out) - liniowa struktura danych, w której dane odkładane są na wierzch stosu i z wierzchołka stosu są zdejmowane. Ideę stosu danych można zilustrować jako stos położonych jedna na drugiej książek – nowy egzemplarz kładzie się na wierzch stosu i z wierzchu stosu 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)

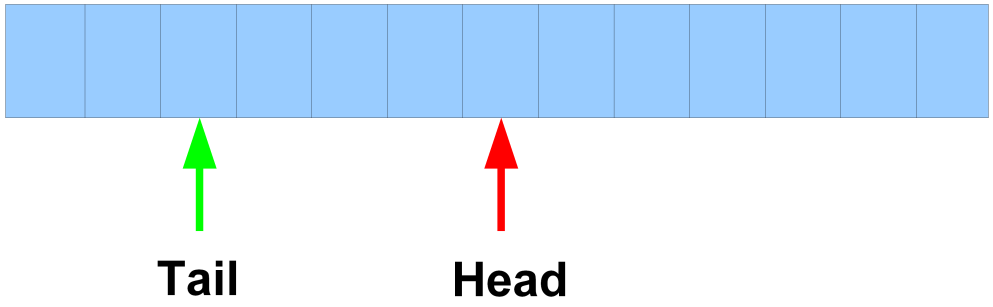


- Dane do kolejki FIFO mogą być wpisywane przez kilka niezależnych aplikacji, wątków lub urządzeń. W takiej sytuacji dostęp do kolejki kontrolowany jest przez Semafor (zmienna globalna).
- Dane zgomadzone w kolejce wysyłane są w kolejności w jakiej zostały wpisane.

Kolejka FIFO (2)

Dane w kolejce FIFO

Adres w pamięci: 0xffD50 0xffD50 + size -1



Zapis danej do kolejki FIFO:

- Zwiększ wskaźnik Head o jeden, zapisz daną.

Odczyt danej z kolejki FIFO:

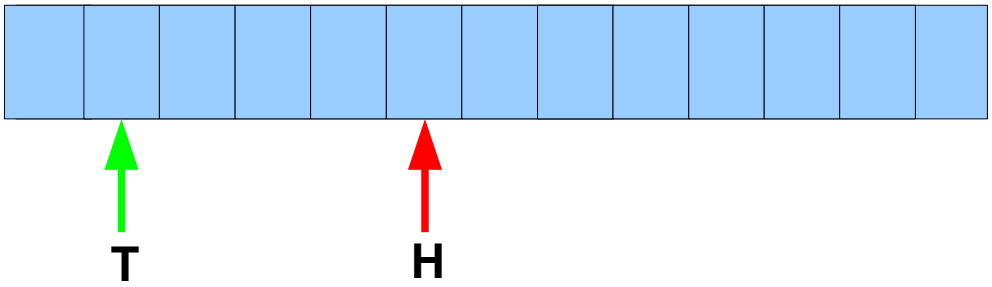
- Odczytaj daną, zwiększ wskaźnik Tail o 1.

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

Kolejka FIFO (3)

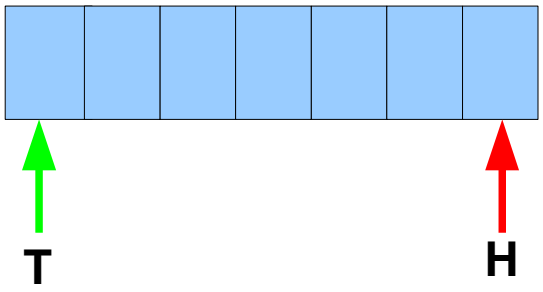
Kolejka pusta T = H

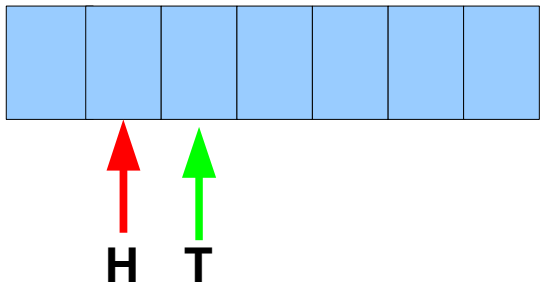
Dane w kolejce, ilość danych = H – T



Brak miejsca w kolejce

(T = 0) & (H = Size) lub T – H = 1





Kolejka FIFO – implementacja w C (1)

```
#define BUFFERSIZE   0xFF                    /* FIFO buffer size and mask */
typedef struct FIFO {
    char buffer [BUFFERSIZE+1];
    unsigned int head;                       // ostrożnie, liczby bez znaku
    unsigned int tail;                       // ostrożnie, liczby bez znaku
};
void FIFO_Init (struct FIFO *Fifo);
void FIFO_Empty (struct FIFO *Fifo);
int FIFO_Put (struct FIFO *Fifo, char Data);
int FIFO_Get (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;          // duża wartość jeżeli tail losowy          /* 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 + 1) & BUFFERSIZE;
        return 1;                                                                    /* Put 1 byte successfully */
}
int FIFO_Get (struct FIFO *Fifo, char *Data){
        If ((TxFifo.head!=TxFifo.tail)){
        *Data = Fifo->buffer[Fifo->tail];
        Fifo->tail = (Fifo->tail + 1) &BUFFERSIZE;
        return 1;                                                                    /* Get 1 byte successfully */
        } else return -1;                                                            /* No data in FIFO */
}
```

Kolejka FIFO – pułapka

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

Enkoder obrotowy

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Enkoder obrotowy

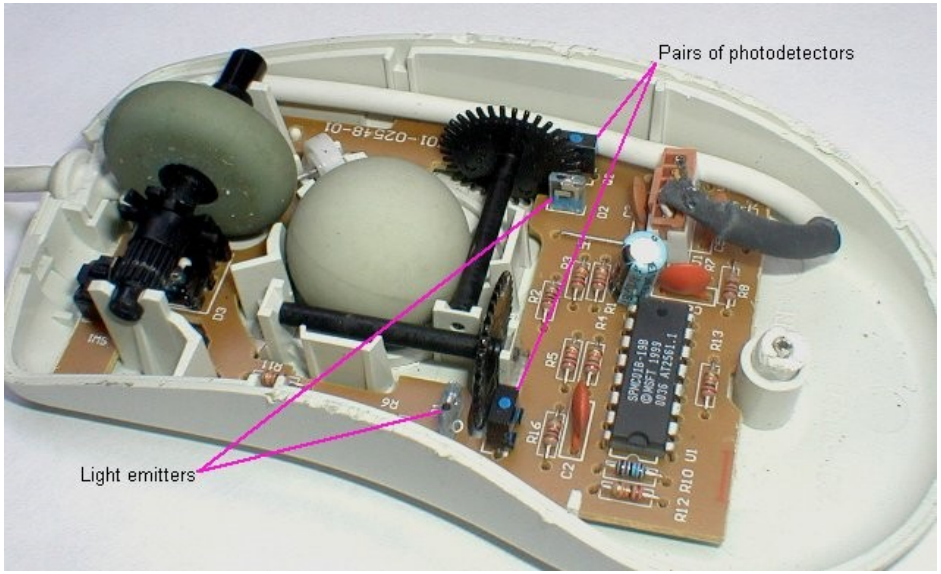
- Enkoder obrotowy (ang. rotary lub shaft encoder) jest elektromechanicznym urządzeniem wykorzystywanym do konwersji położenia kątowego do sygnału analogowego lub cyfrowego. Enkodery obrotowe wykorzystywane są w różnych aplikacjach wymagających precyzyjnego określenia położenia kątowego lub kierunku obrotu,np. systemy sterowania, robotyka, specjalizowane obiektywy fotograficzne, urządzenia wejściowe do urządzeń komputerowych (myszki, tackball-e) oraz urządzenia radarowe.
- Wyróżniamy dwa rodzaje enkoderów podające położenie:
  - Bezwzględne,
  - Względne (przyrostowe).
- Enkodery bezwzględne generują unikalny kod cyfrowy dla różnych kątów obrotu.
- Enkodery przyrostowe, znane jako kwadraturowe, posiadają zwykle dwa wyprowadzenia. Do określenia zmiany kąta położenia wykorzystuje się różnicę faz wygenerowanych sygnałów.
- Enkodery mogą być mechaniczne lub optyczne. Optyczne enkodery wykorzystują cyfrowe mozaiki oparte na kodzie Graya.

Przykłady enkoderów

Precyzyjny pomiar przesunięcia, kąta obrotu

Potencjometry cyfrowe

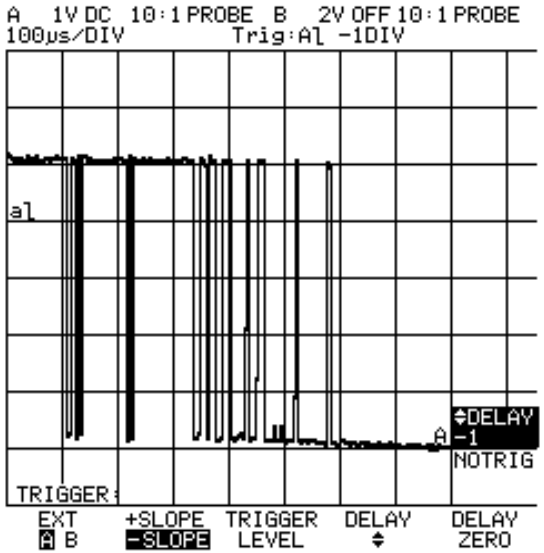
Myszka komputerowa i enkodery optyczne

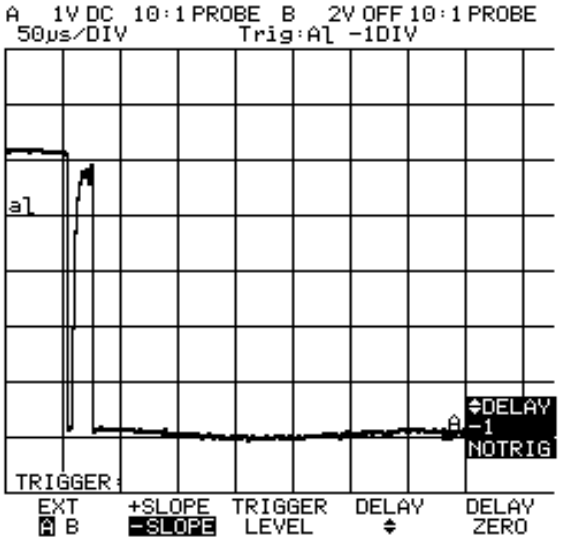


Enkodery kwadraturowe (1)

- Enkodery mechaniczne wymagają filtracji drgań styków mechanicznych. Ze względu na niską cenę wykorzystywane są jako potencjometry cyfrowe w sprzęcie audio (potencjometr regulujący głośność, sterowanie menu). Ze względu na drgania styków enkodery mechaniczne mają stosunkowo niską szybkość obrotową.

Drgania styków enkodera mechanicznego



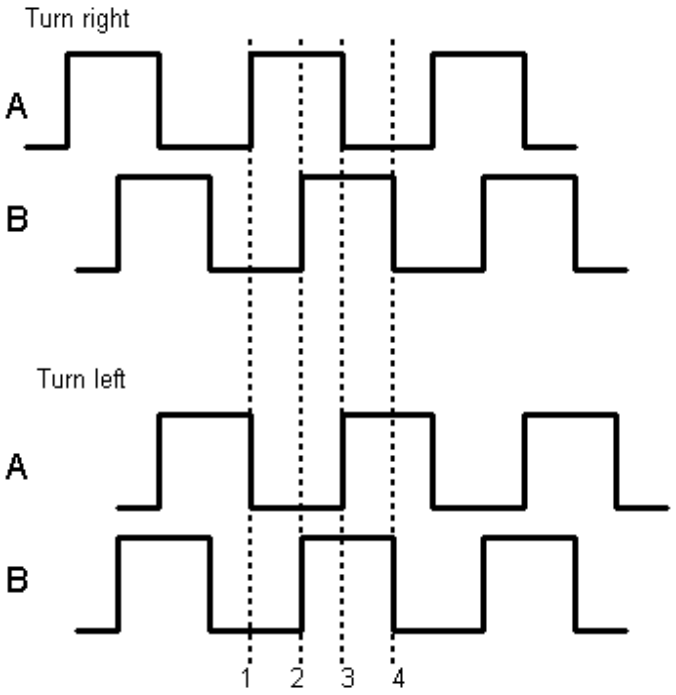


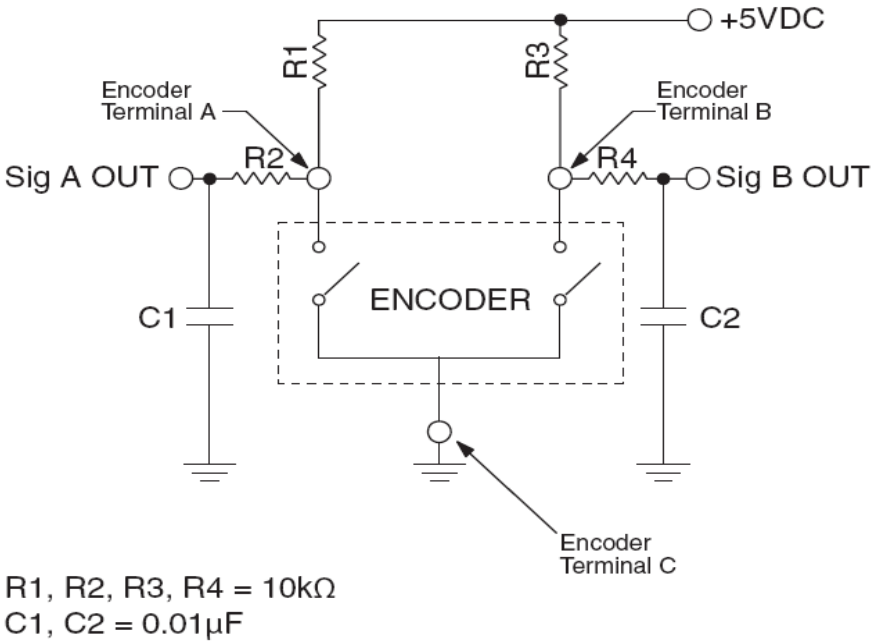
Electrical
MAXIMUM VOLTAGE: 5 VDC.
MAXIMUM CURRENT DC: 0,5 mA
DIELECTRIC STRENGTH: 50 VAC.
INSULATION RESISTANCE: 10 MΩ.
BOUNCE TIME: ≤ 2 ms

Electrical
MAXIMUM VOLTAGE: 5 VDC.
CURRENT DC: 1 mA
DIELECTRIC STRENGTH: 50 VAC.
INSULATION RESISTANCE: 50 MΩ min / 50 VDC.
BOUNCE TIME: ≤ 5 ms

Enkodery kwadraturowe (2)

- Różnica faz sygnałów generowanych przez enkodery cyfrowe (sygnały A i B) pozwala na określenie kierunku obrotu. Częstotliwość generowanego przebiegu prostokątnego pozwala na określenie prędkości kątowej.

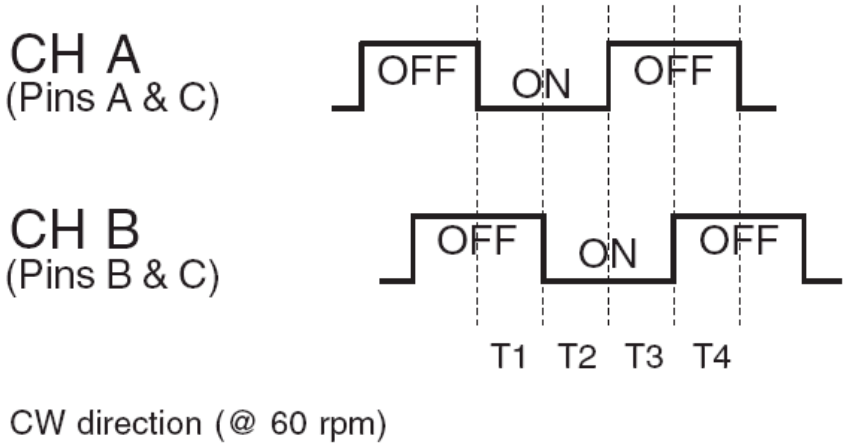




Analogowy układ filtrujący drgania styków

Enkodery kwadraturowe (3)

- Wyjścia enkodera generują sygnał kwadraturowy – różnica faz obu sygnałów wynosi 90 stopni. Odfiltrowane sygnały wykorzystywane są do generowania impulsów inkrementujących lub dekrementujących licznik. W systemach mikroprocesorowych wykorzystuje się wyzwalanie dowolnym zboczem lub poziomem sygnału generowanego przez enkoder. W celu wyznaczenia kierunku obrotu należy zaimplementować maszynę stanową, jeżeli poprzednim stanem było “00”, a obecny stan wynosi “01” to enkoder obracany jest zgodnie ze wskazówkami zegara. Drgania styków mogą spowodować niewłaściwe działanie automatu stanowego, np. przejście 00->11 uniemożliwia określenie kierunku obrotu, mamy dwie możliwości: 00->01->11 oraz 00->10->11.

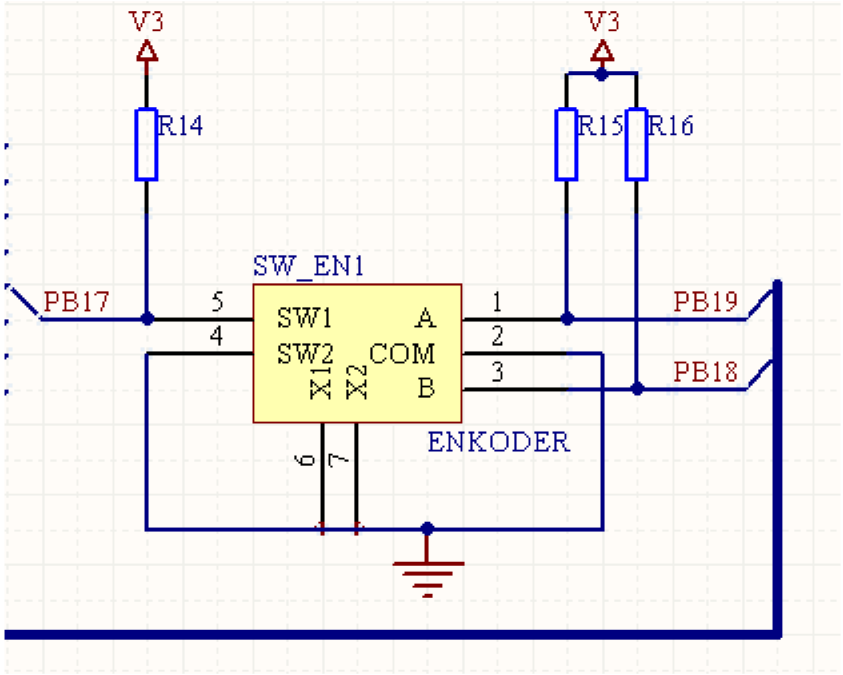


Phase	A	B
1	0	0
2	0	1
3	1	1
4	1	0

Phase	A	B
1	1	0
2	1	1
3	0	1
4	0	0

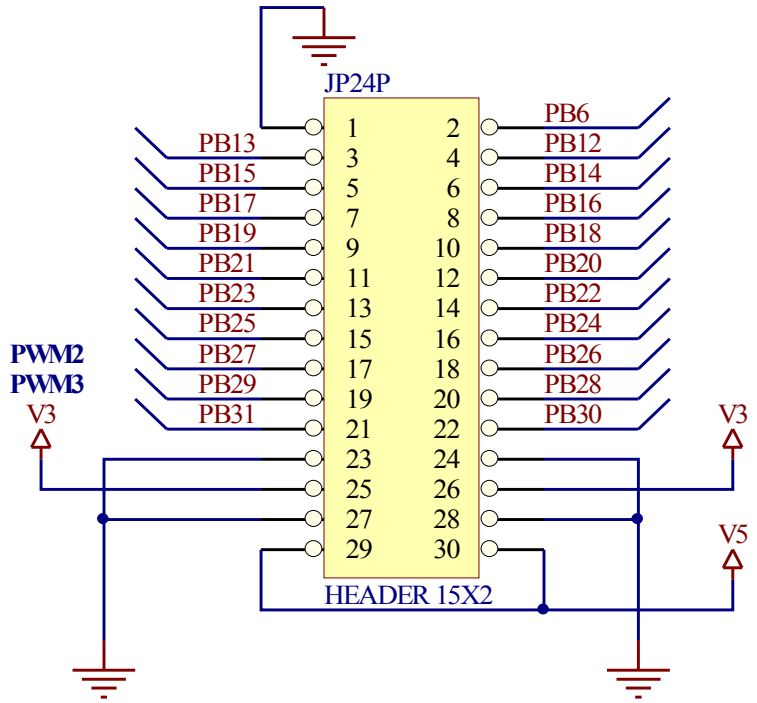
Enkodery kwadraturowe – laboratorium

- Schemat elektryczny układu enkodera podłączonego do procesora AMR wykorzystywanego na zajęciach.
  - Enkoder wymaga dobrego algorytmu filtrującego drgania styków.

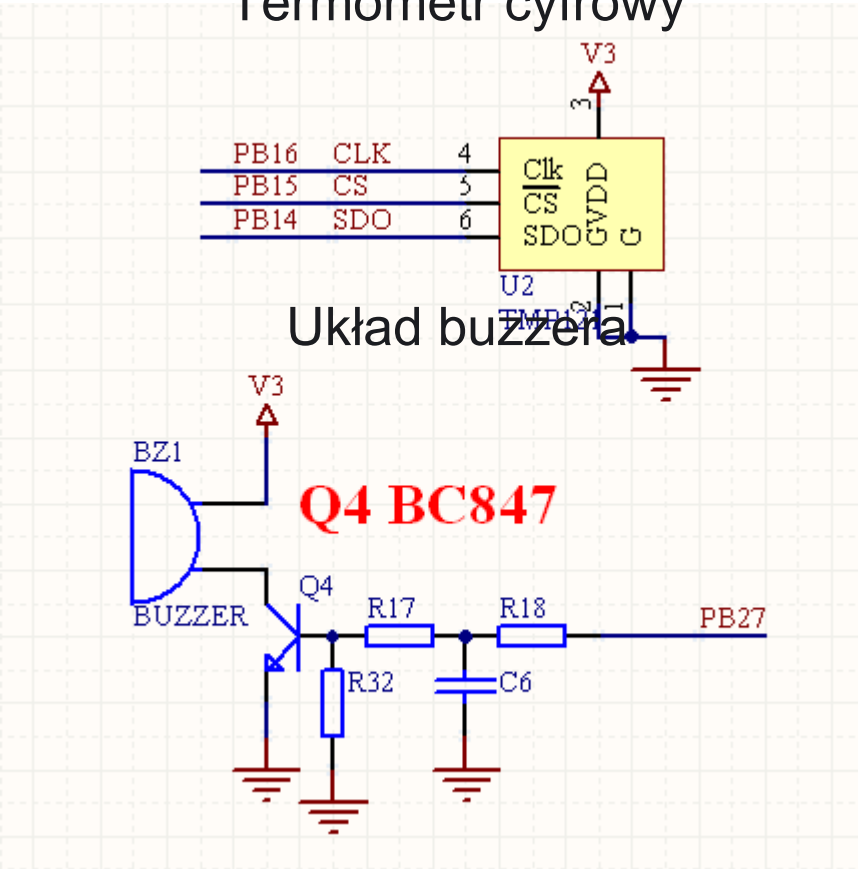


Płyta nakładkowa – termometr, buzzer

Złącze do procesora ARM



Termometr cyfrowy



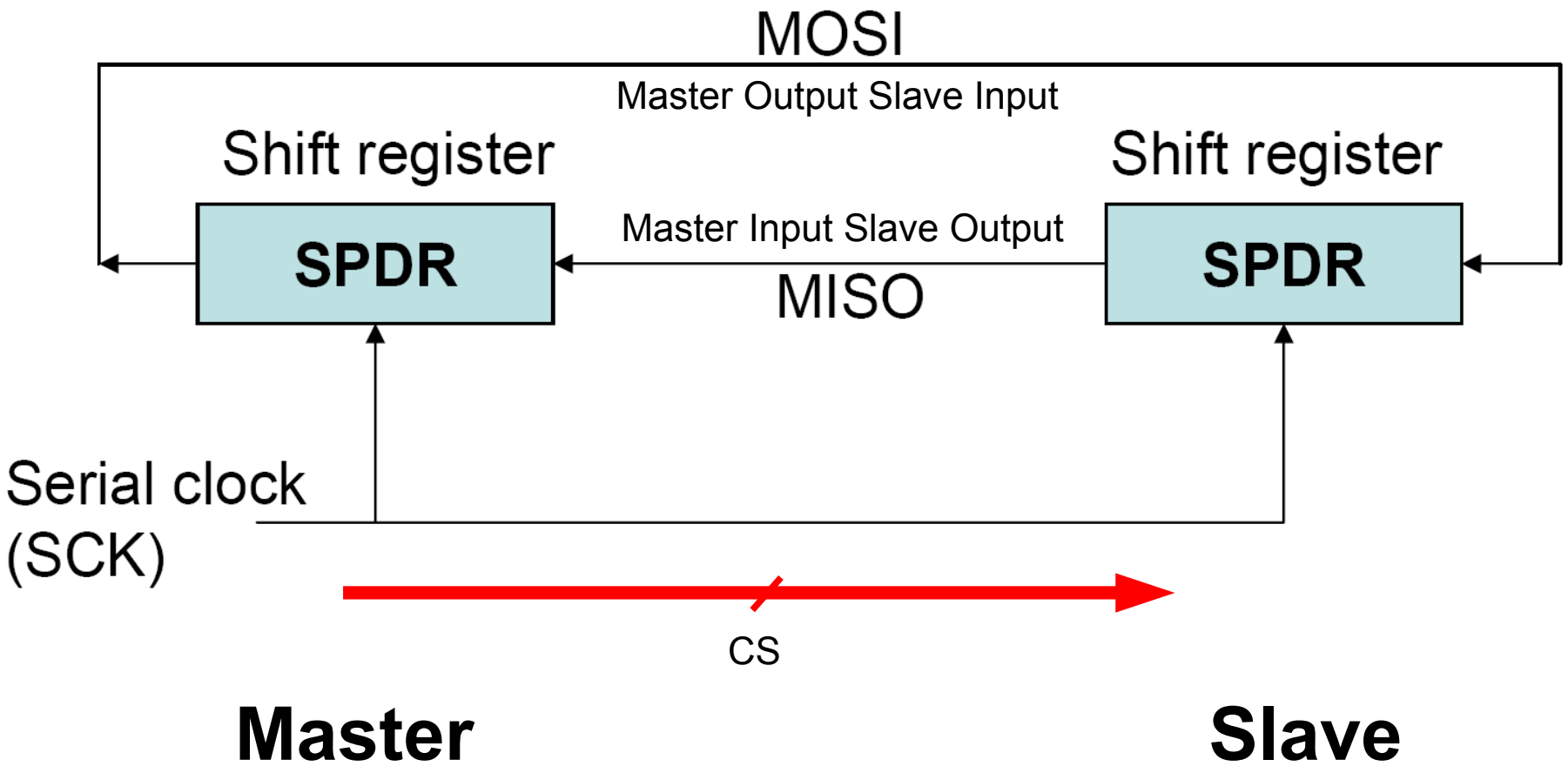
Serial Peripheral Interface

Serial Peripheral Interface

Cechy interfejsu SPI:

- Szeregowa transmisja synchroniczna,
- 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



Protokół interfejsu SPI

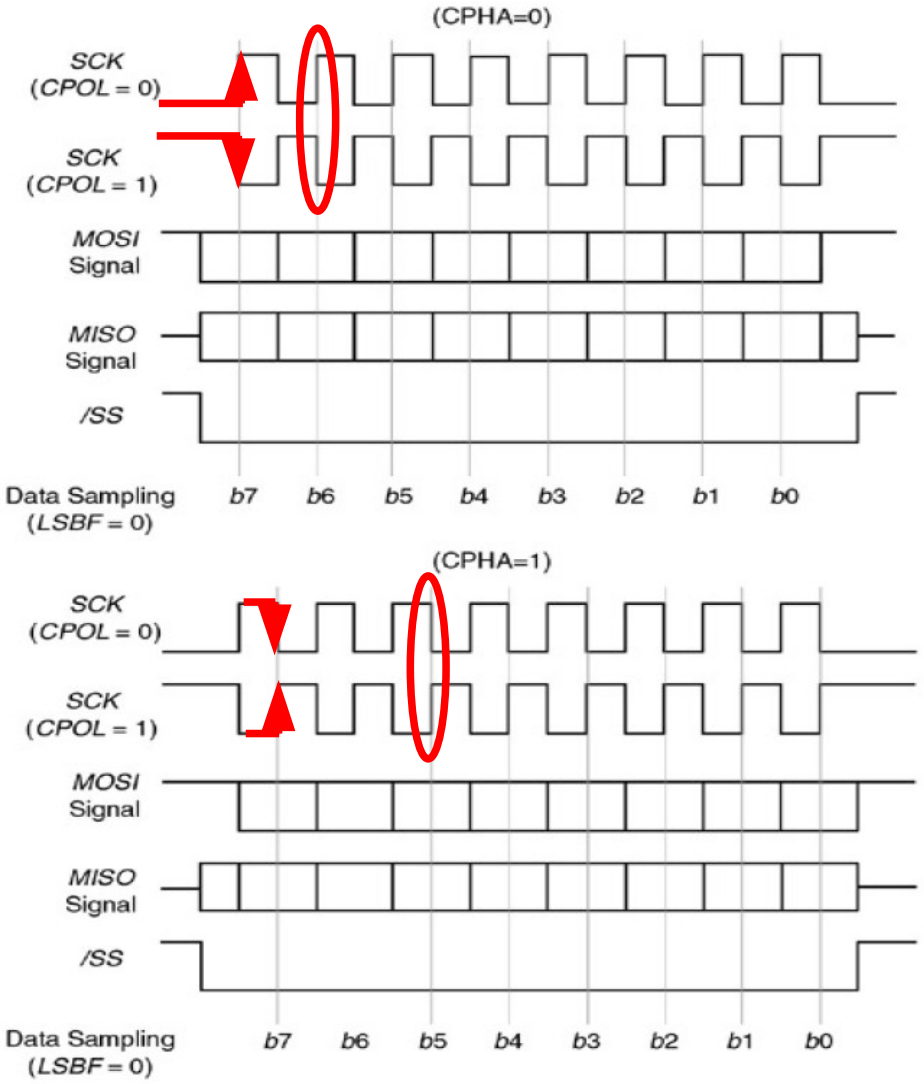
Konfiguracja sygnału zegarowego:

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 (próbkowanie na pierwszym zboczu zegara),

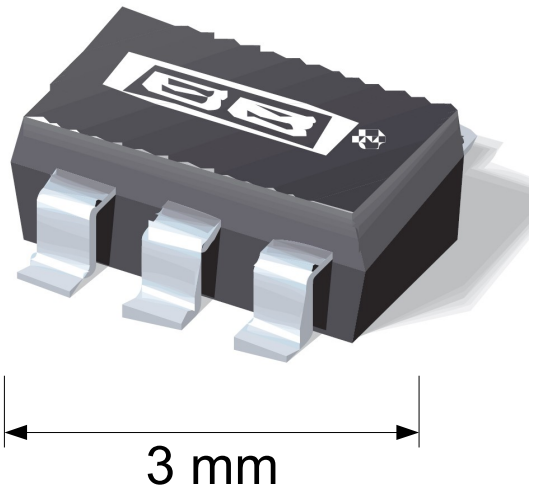
Opóźniona faza zegara (próbkowanie na drugim zboczu zegara).



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: ±1.5°C od −25°C do +85°C
- Pobór prądu w stanie uśpienia: 50µA (mak.)
- Zasilanie: 2,7V to 5,5V



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

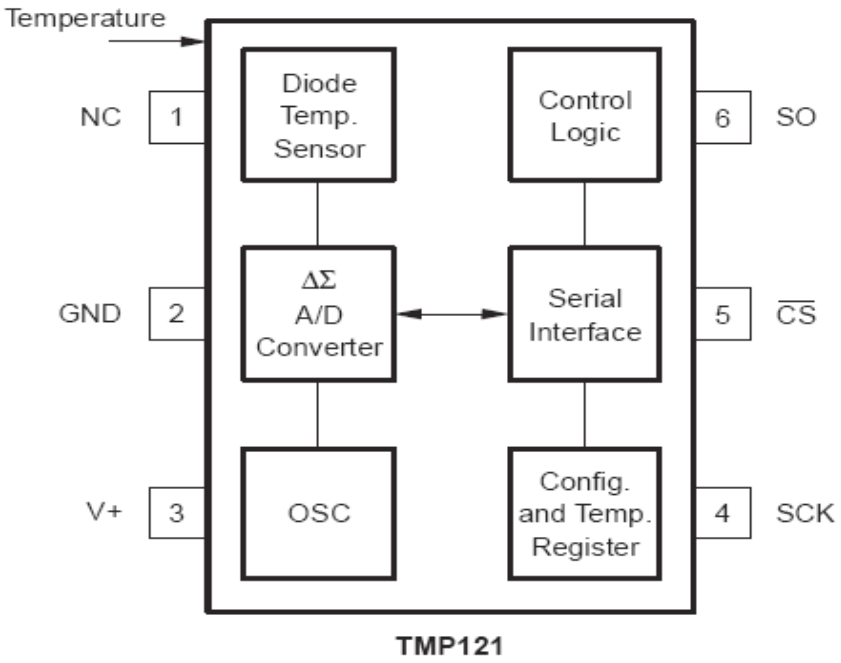
D7	D6	D5	D4	D3	D2	D1	D0
T4	T3	T2	T1	T0	0	Z	Z

Table 1. Temperature Register

TEMPERATURE (°C)	DIGITAL OUTPUT(1) (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
(1) The last two bits are high impedance and are shown as 00 in the table.		

Table 2. Temperature Data Format

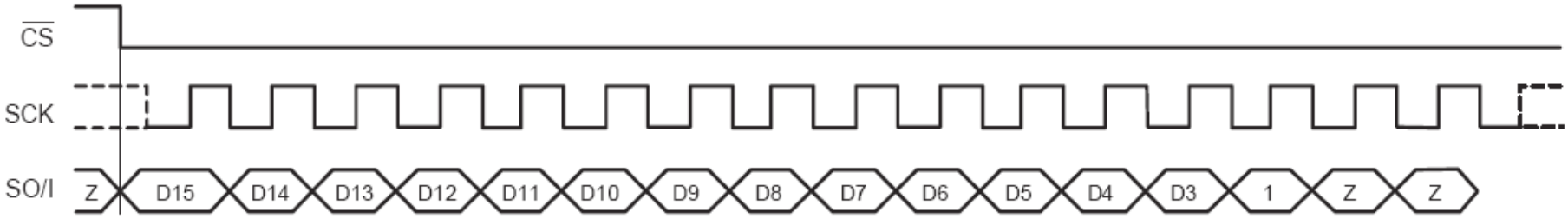
Ramka SPI termometru TMP121



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

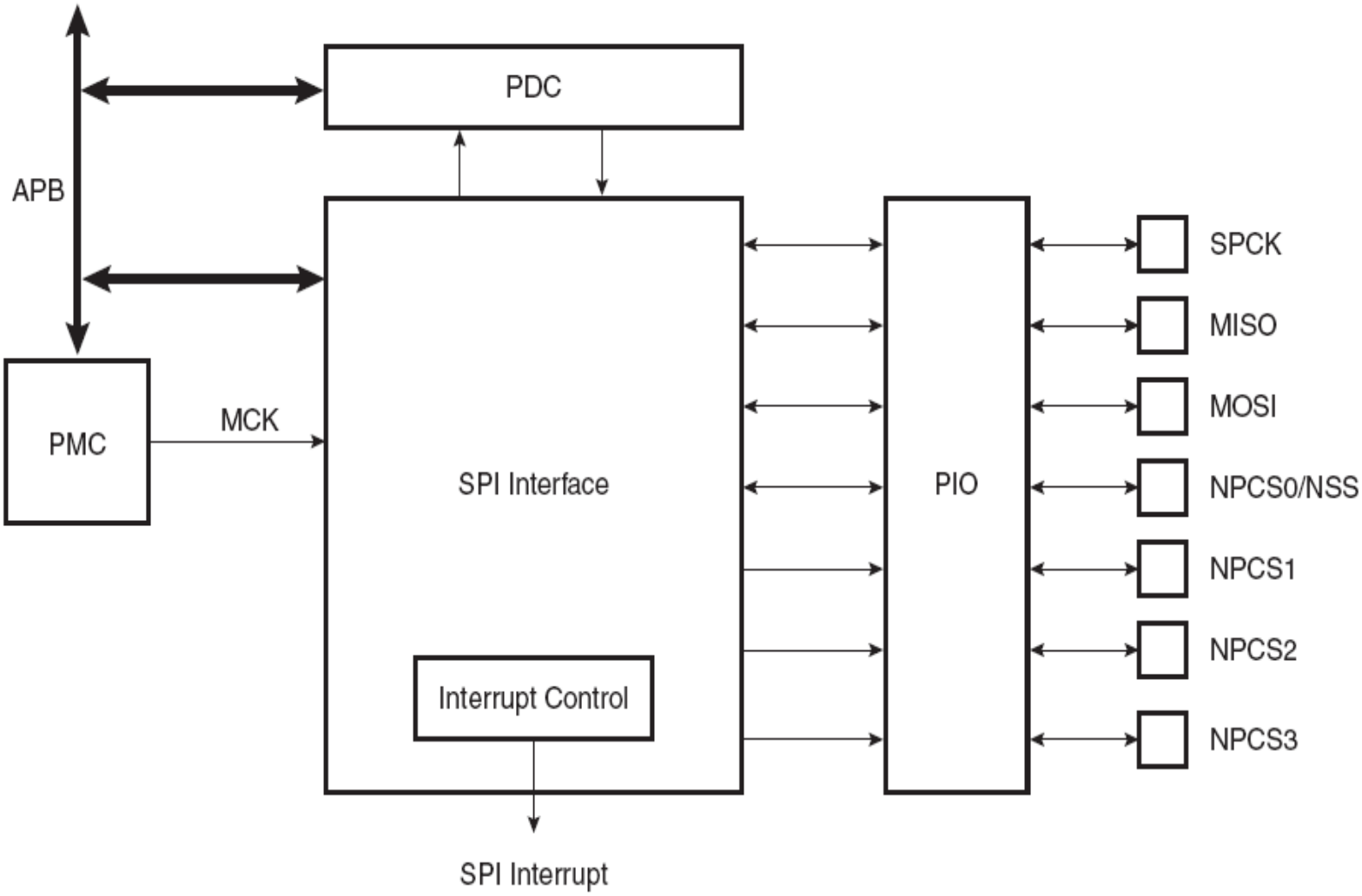


Moduł SPI procesora ARM AT91SAM9263 (1)

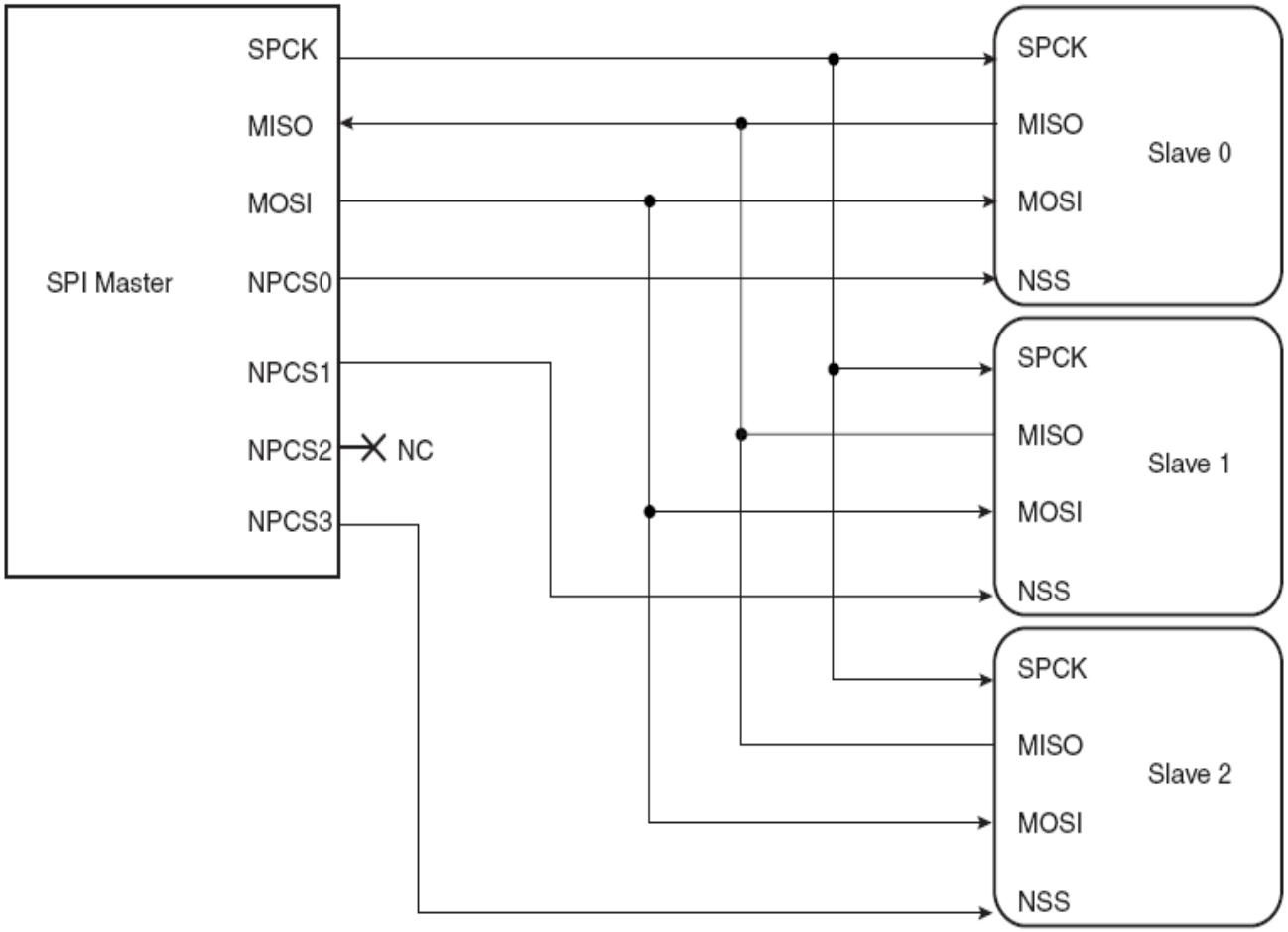
Cechy modułu SPI:

- 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)



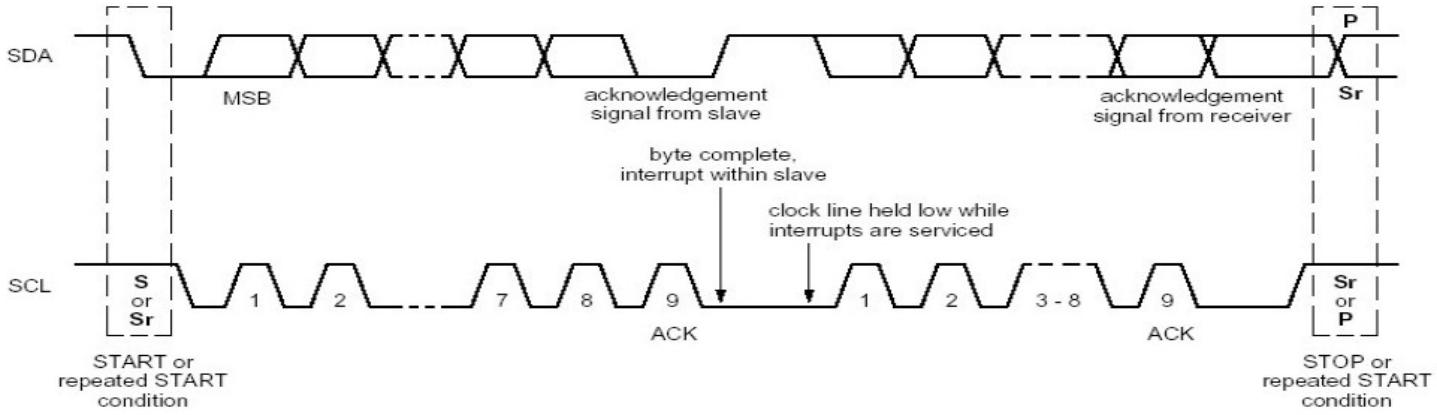
Moduł SPI procesora ARM (3)



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.

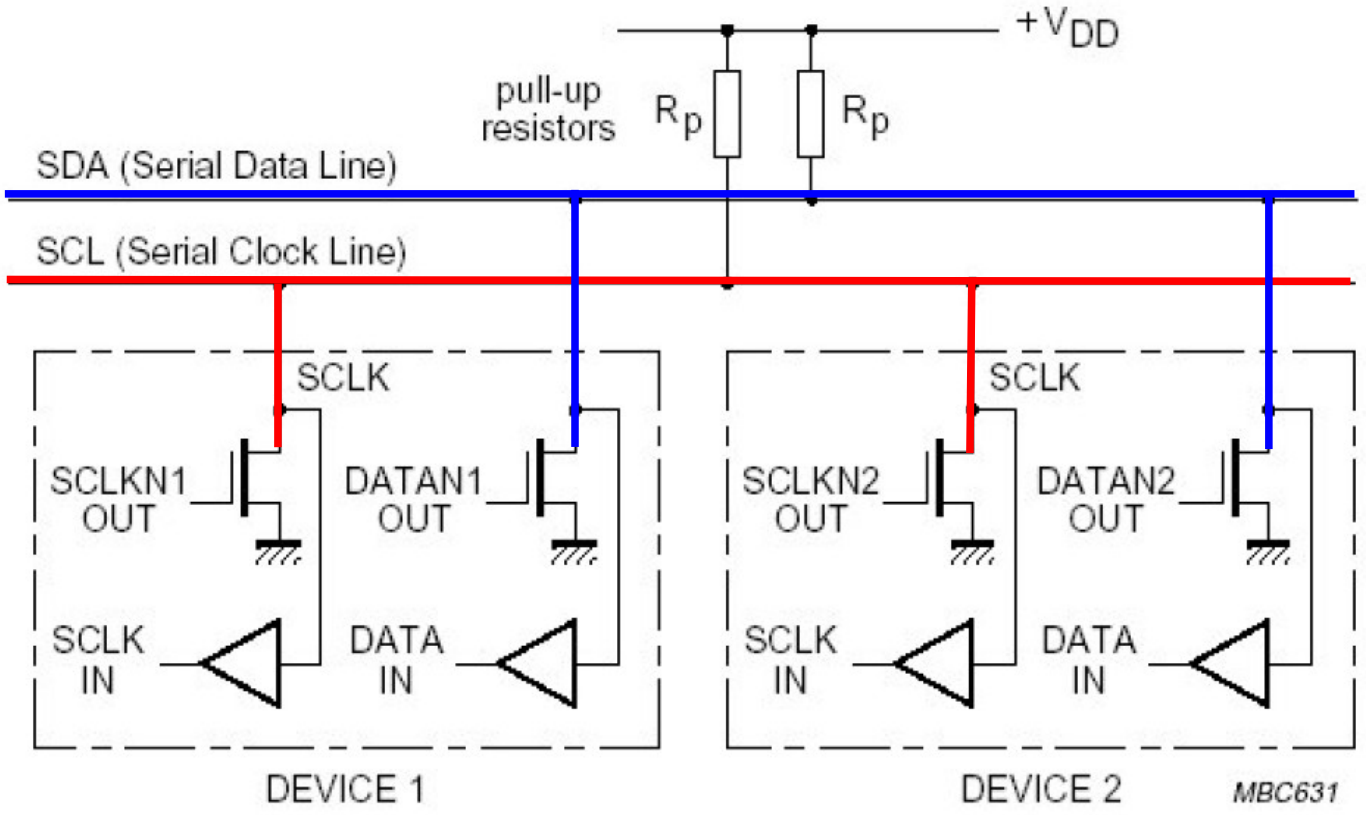


Zastosowanie interfejsu I2C

W sprzedaży dostępnych jest wiele bardzo tanich układów scalonych sterowanych poprzez I²C:

- 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-kanałowy przetwornik analogowo-cyfrowy i cyfrowo-analogowy

Magistrala I2C

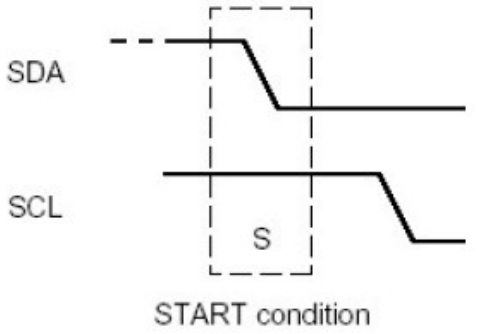


Urządzenie nadrzędne (Master) – inicjuje transmisję, generuje sygnał zegarowy

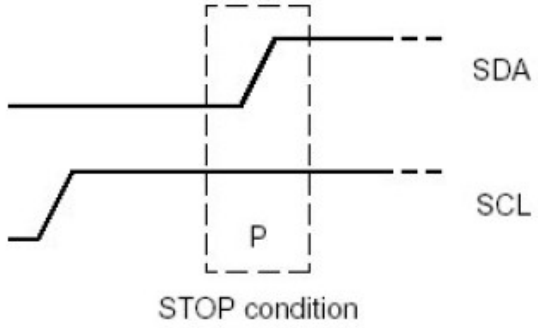
Urządzenie podrzędne (Slave) – analizuje wysłany przez urządzenie adres i transmituje lub odbiera dane.

Rozpoczęcie oraz zakończenie transmisji

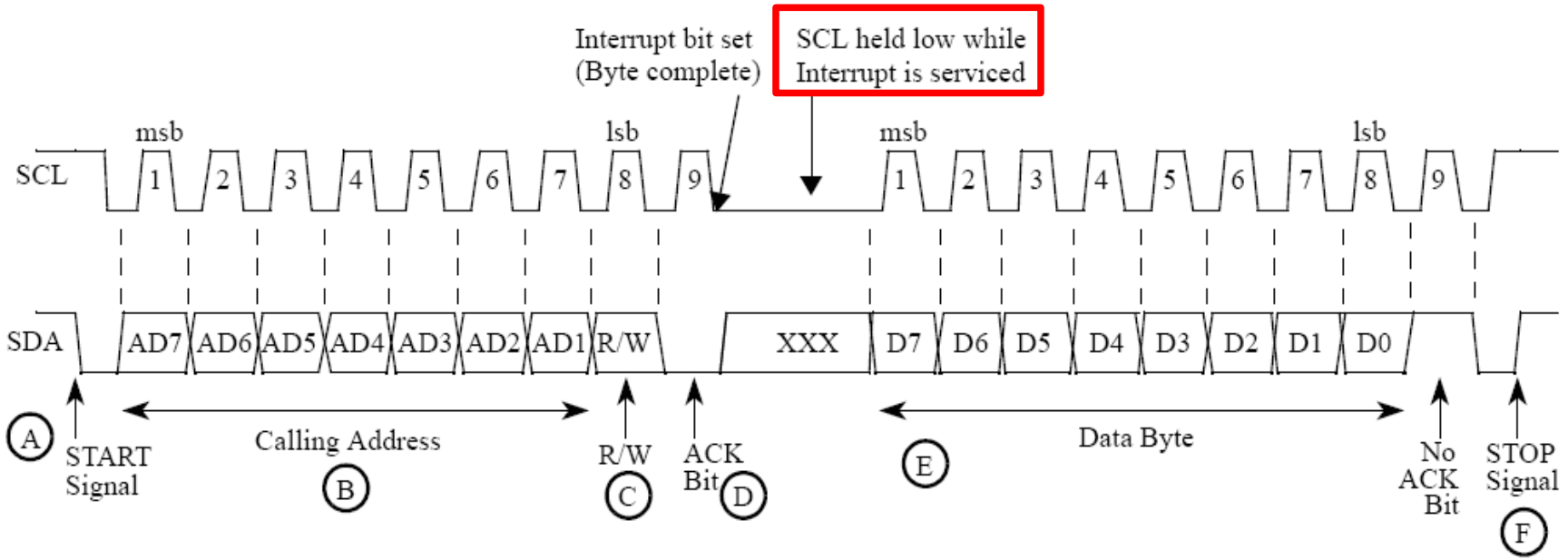
Rozpoczęcie transmisji – generacja sygnału **START** (opadające zbocze na szynie SDA, zmiana stanu z “1” na “0” logiczne, podczas ważnego sygnału SCL = ”1”). Sygnał generuje Master.



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

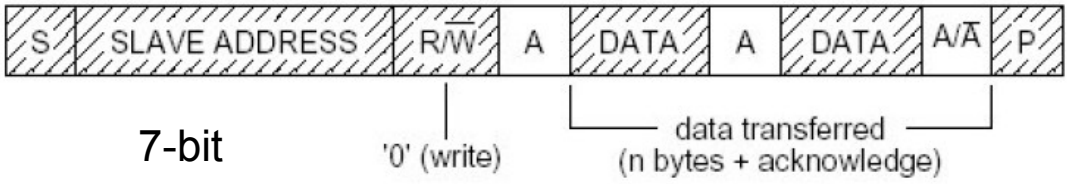


A) Transmisje rozpoczyna Master generując sygnał START.
B) Następnie transmituje 8 bitów danych (7 bitów adresowych, bit R/W).
C) Po transmisji 8 bitów Slave przejmuje magistralę i wymusza odpowiedni poziom na linii SDA (9 takt zegara). Odpowiada w ten sposób bitem potwierdzenia ACK (brak potwierdzenia, 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 lub odczyt

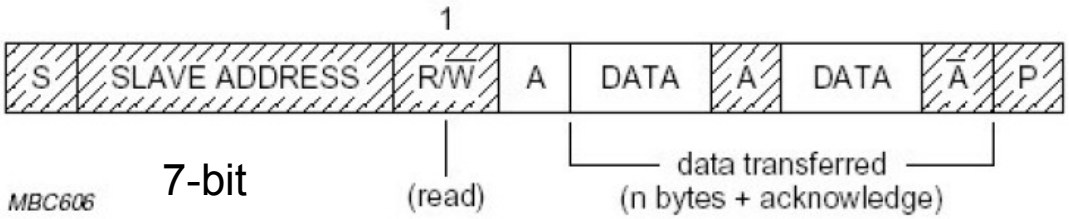
Zapis n-bajtów danych

master-transmitter



Odczyt n-bajtów danych

master-receiver (since second byte)



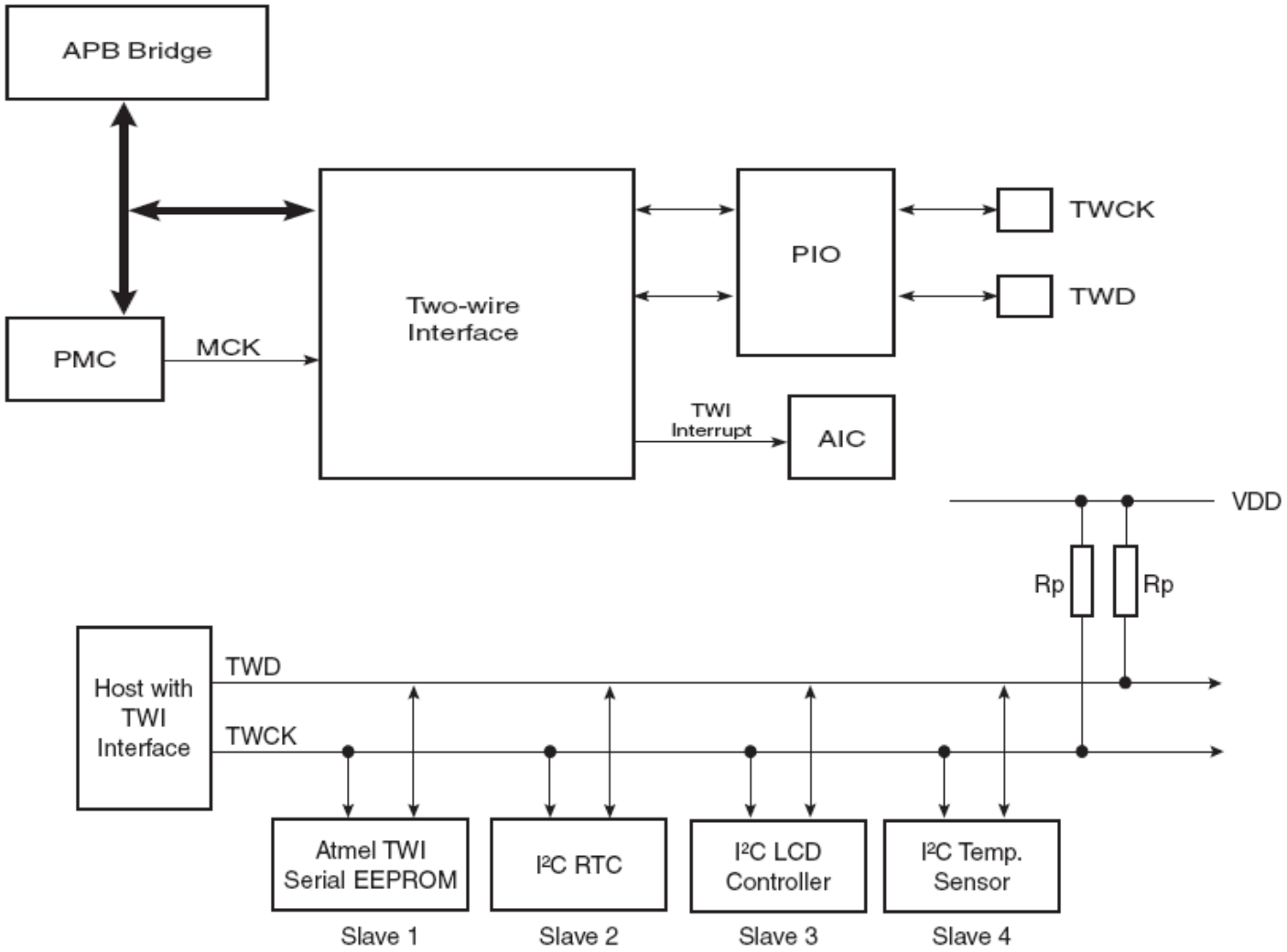
Two-Wire Interface – standard zgodny z I2C ?

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 przerwaniami,
- 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.

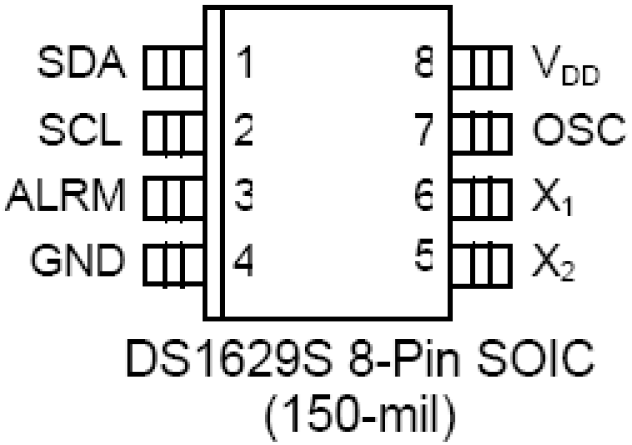
Schemat blokowy modułu TWI



Zegar czasu rzeczywistego

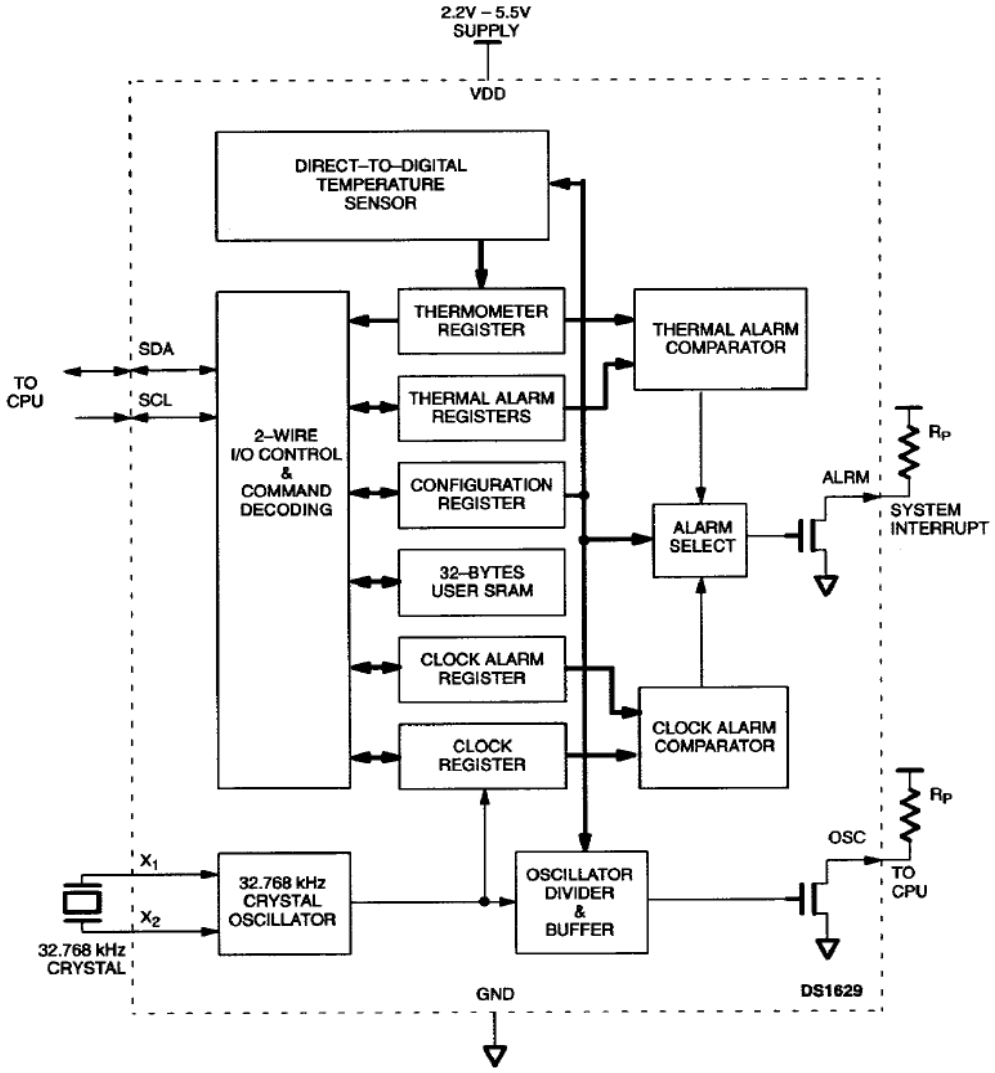
Cechy układu DS1629:

- 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).

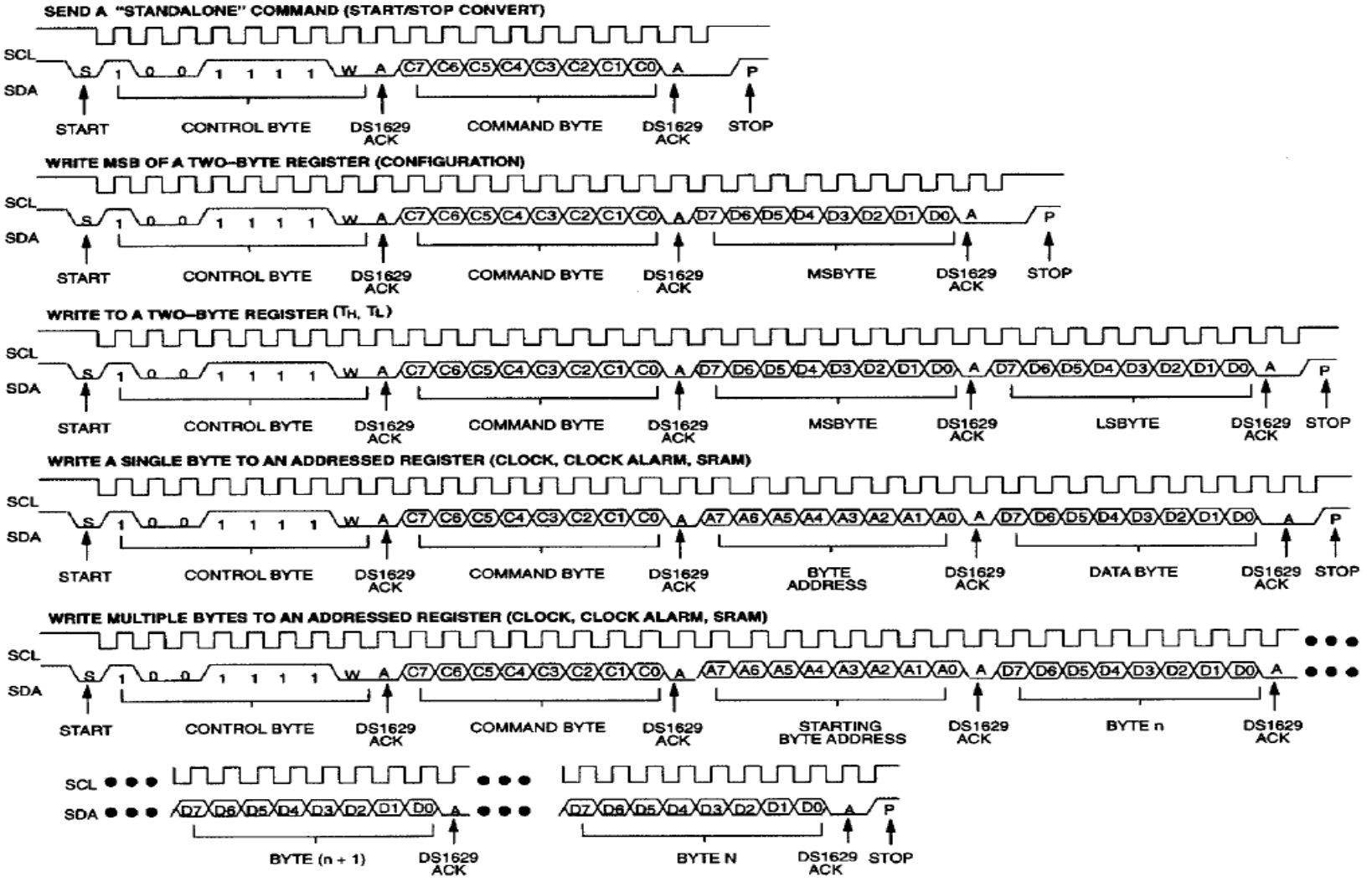


DS1629S 8-Pin SOIC
(150-mil)

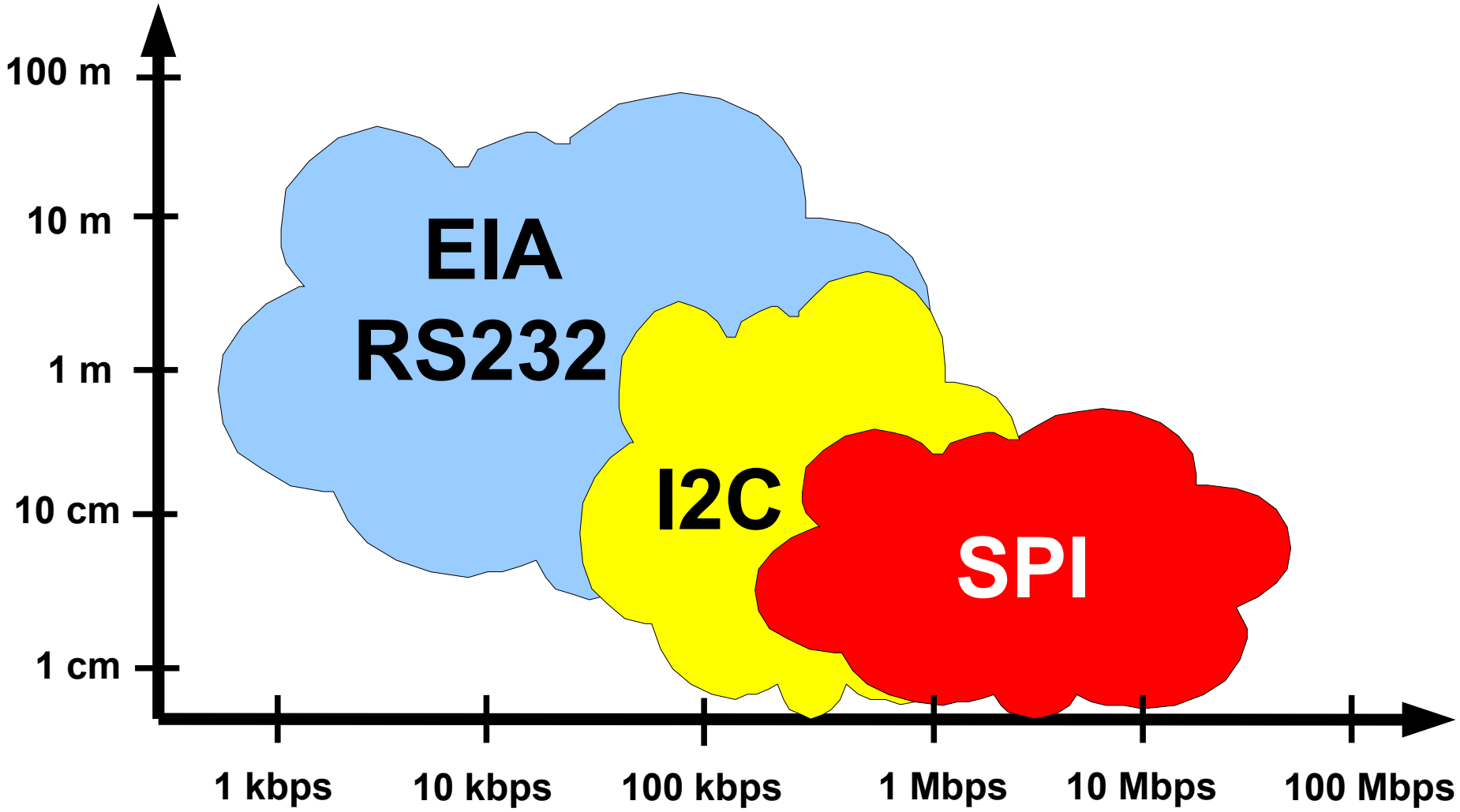
Zegar czasu rzeczywistego



Transmisja z wykorzystaniem interfejsu I2C

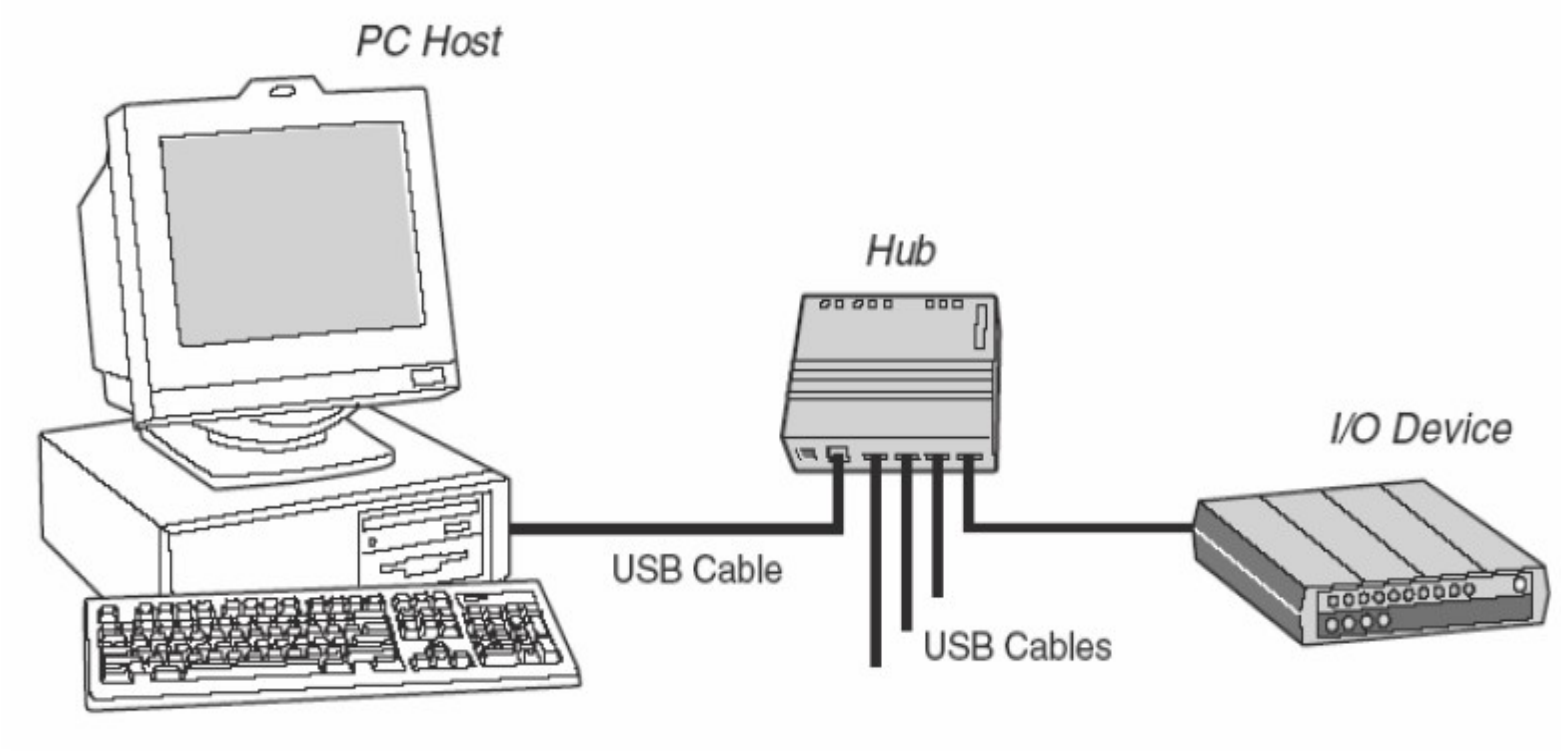


Interfejsy szeregowe - podsumowanie



Magistrala USB
(Universal Serial Bus)

Magistrala USB

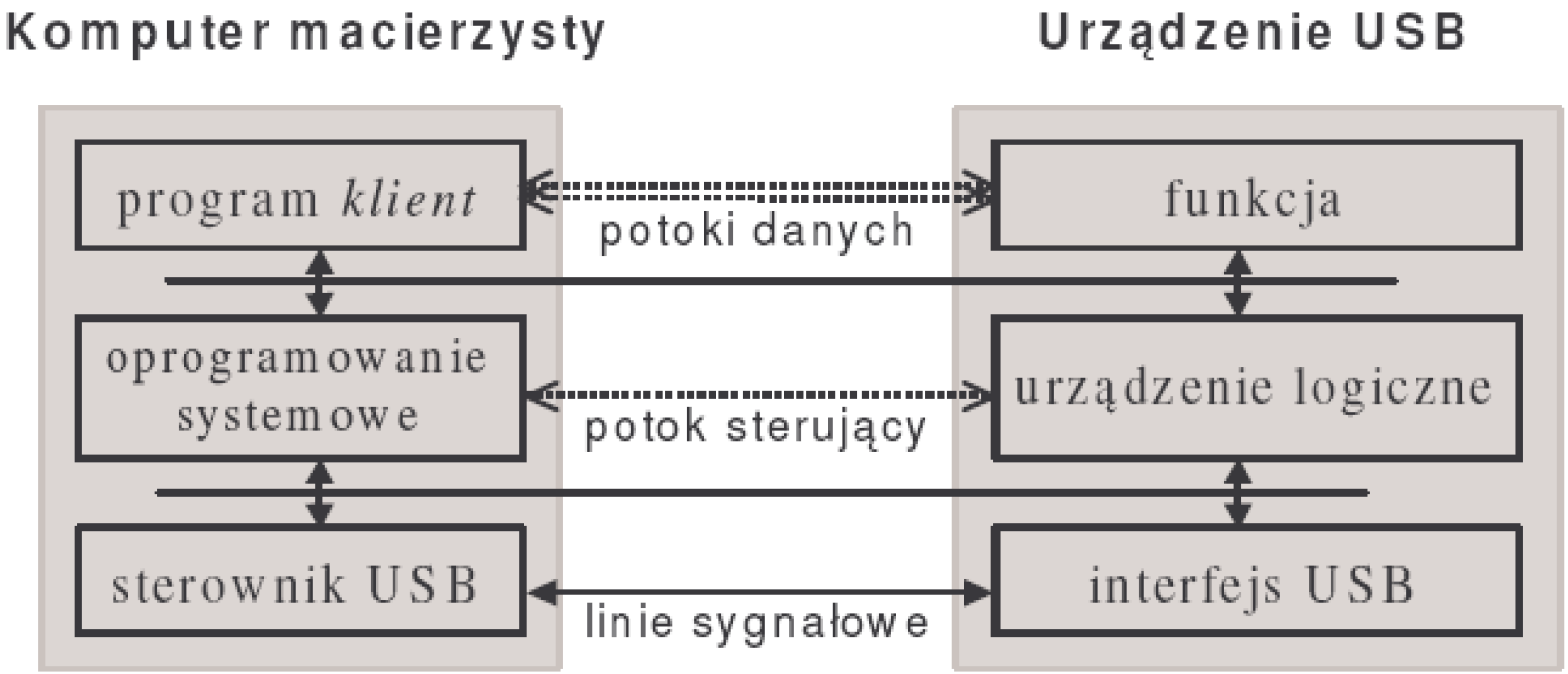


Cechy magistrali USB

- Asynchroniczna, szeregowa, 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.**

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

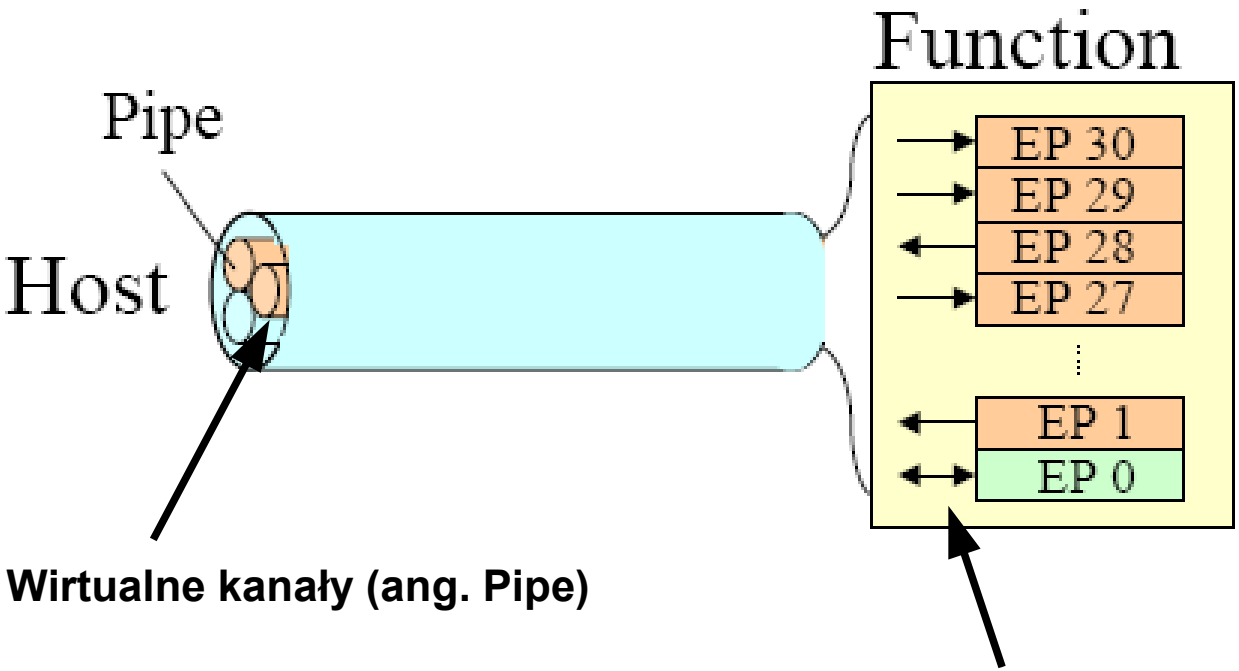
Struktura warstwowa magistrali USB



Magistrala USB zbudowana jest na bazie architektury typu gwiazda.
Model systemu USB składa się z trzech warstw:

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

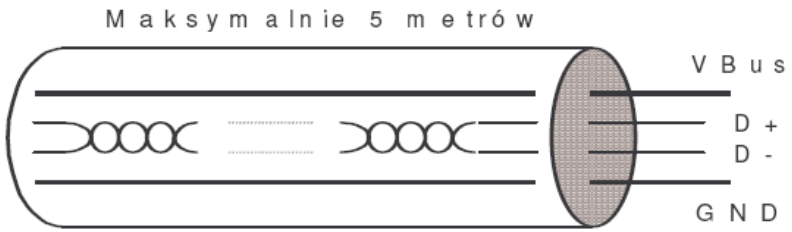
Przepływ danych w systemie USB



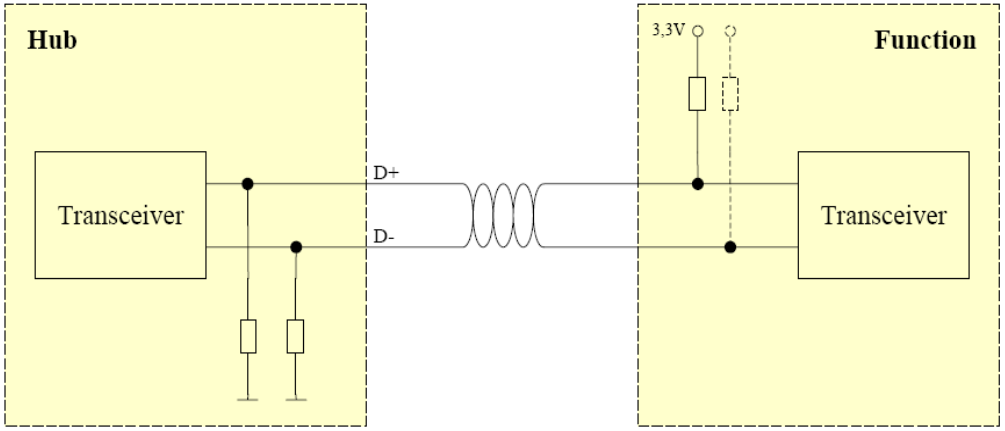
Wirtualne kanały (ang. Pipe)

Kanał sterujący (EP0) oraz
Kanały danych EP1 – EP30 (ang. End Points)

Warstwa fizyczna



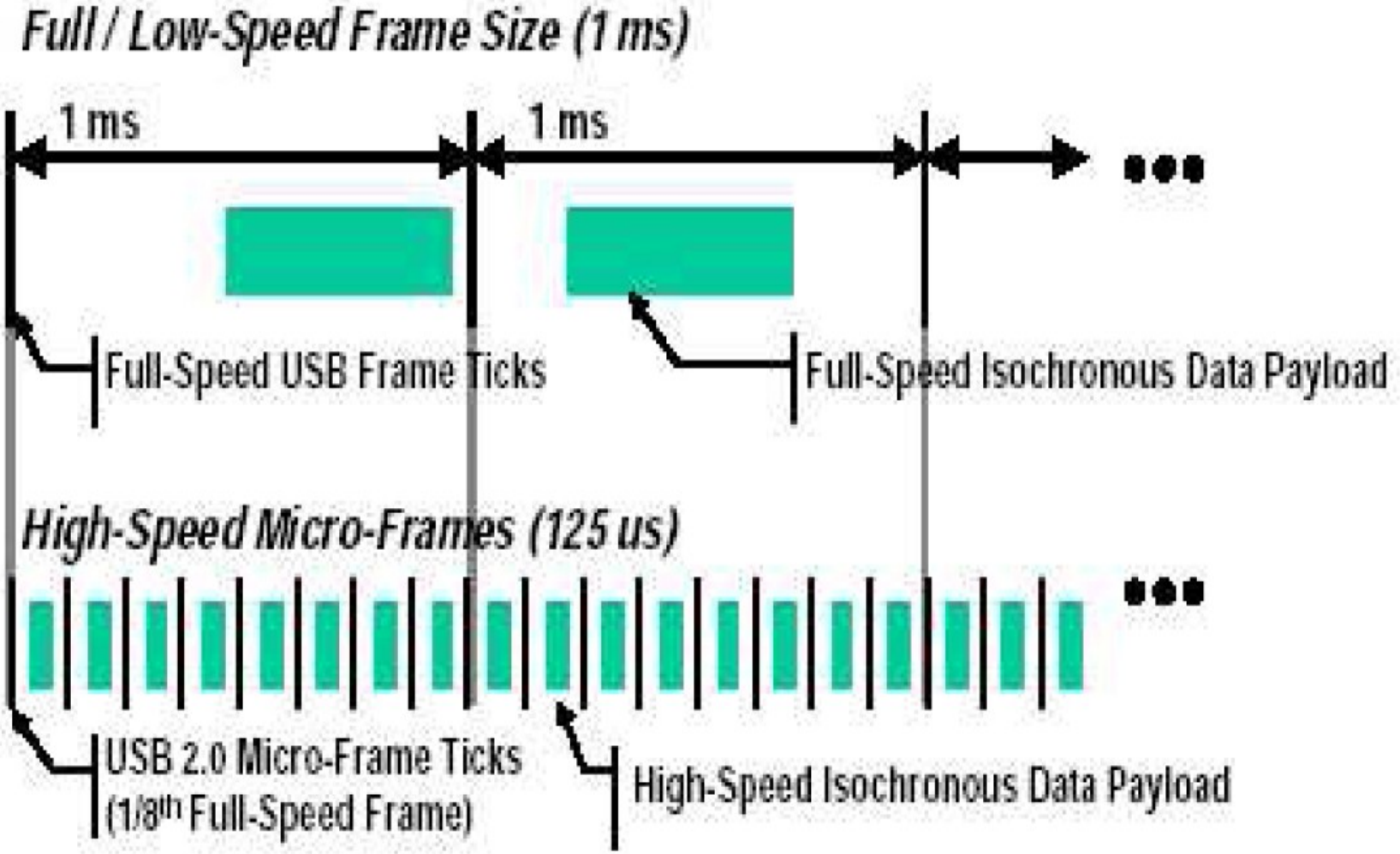
Transmisja różnicowa, typu half-duplex. Dwa dodatkowe przewody zasilające 5 V/500 mA



Złącza typu mini USB

Złącza USB typu “A” i “B”

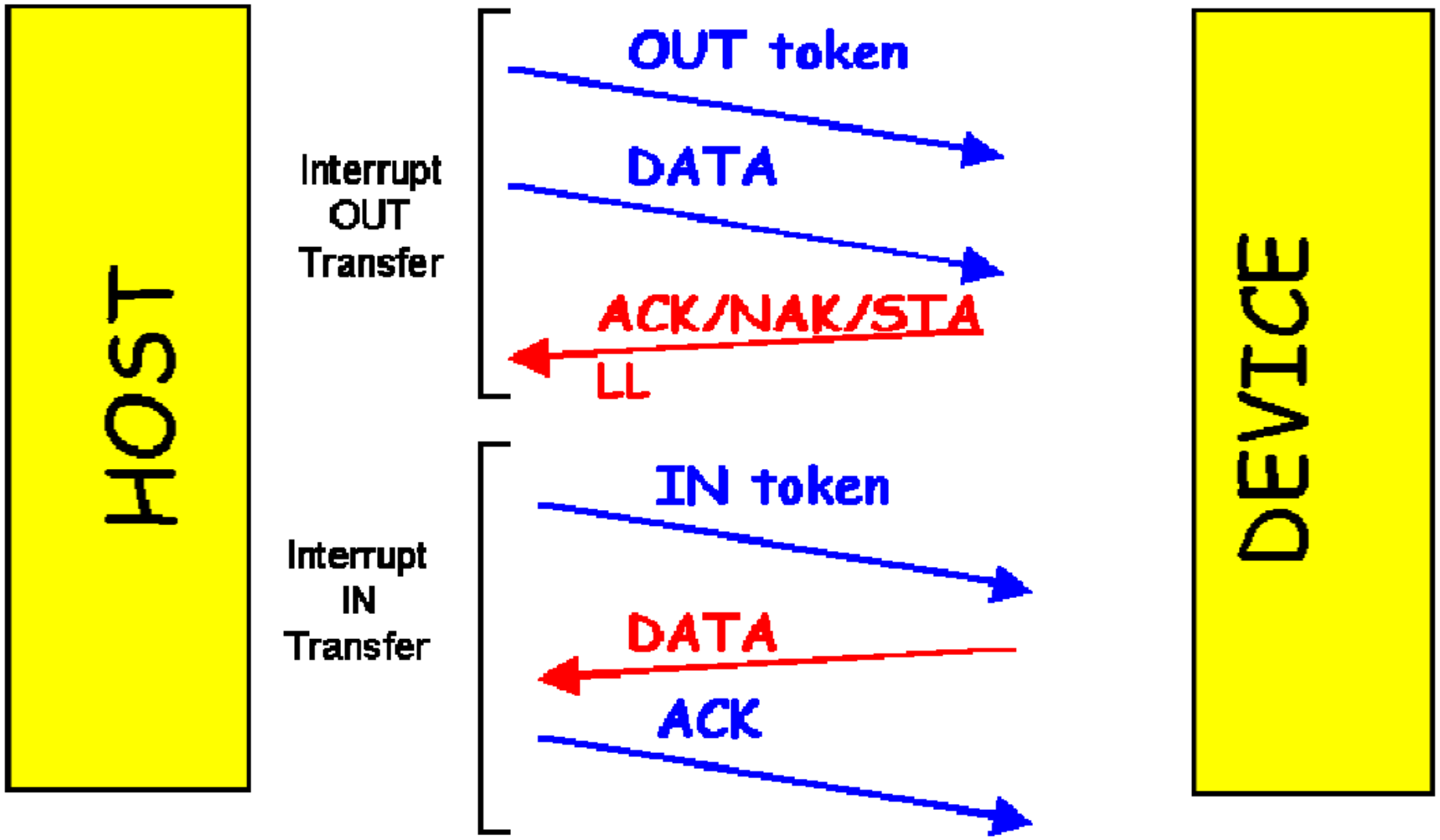
Zależności czasowe ramek USB



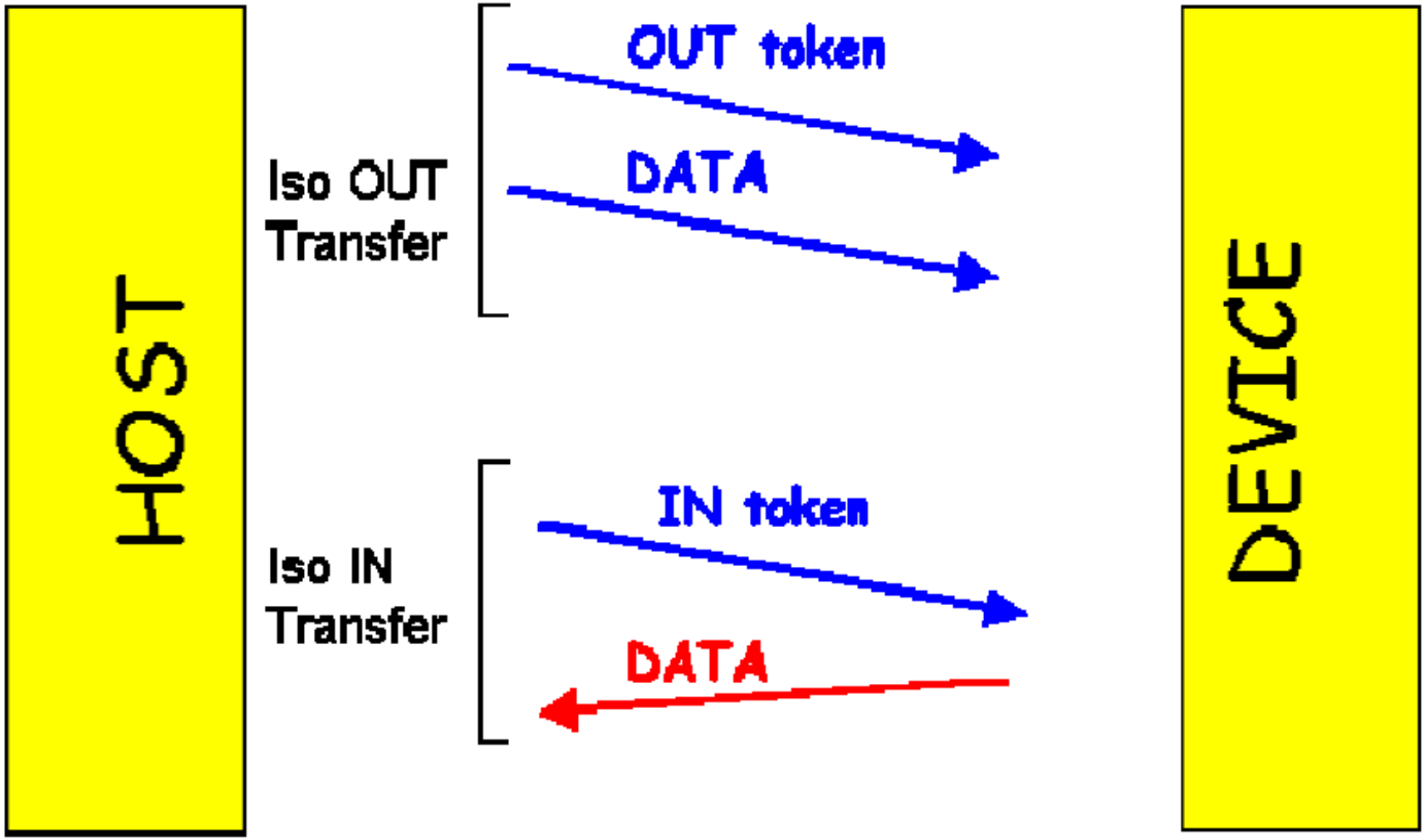
Rodzaje transferów

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

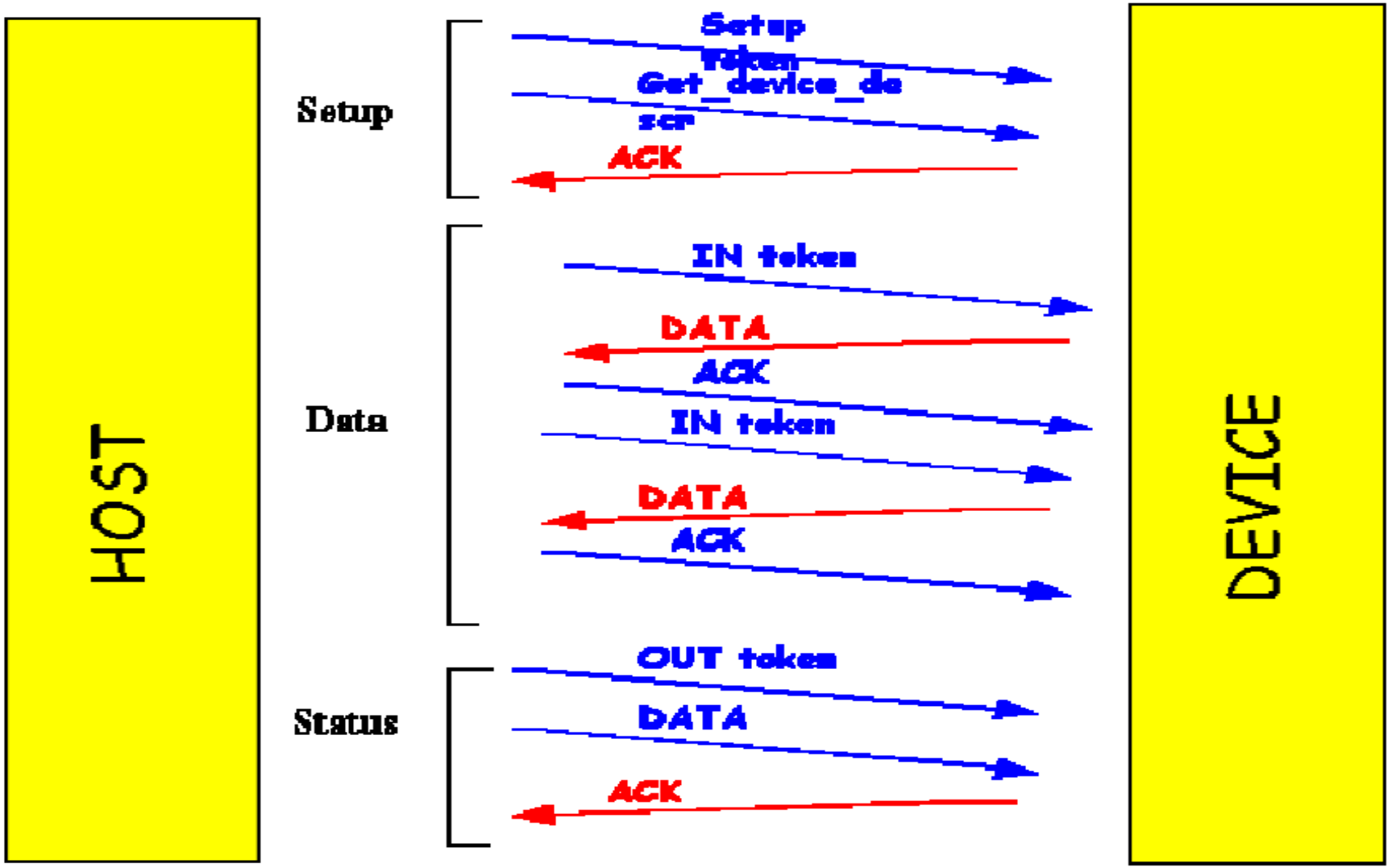
Transfer przerwaniowy i masowy



Transfer izochroniczny



Transfer sterujący



Proces konfiguracji

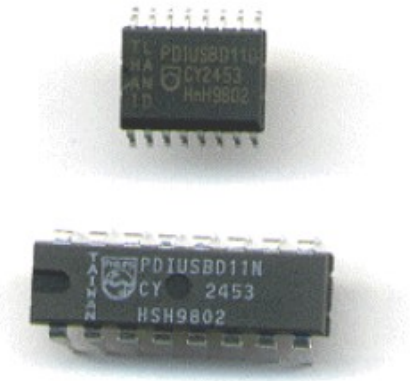
Enumeracja (ang. Enumeration) – konfiguracja urządzeń przeprowadzana po dołączeniu lub odłączeniu nowego urządzenia od magistrali. Proces konfiguracji przeprowadzany jest przez urządzenie nadrzędne (Master). Master przypisuje indywidualne adresy do urządzeń oraz ustanawia podstawowe parametry transmisji:

- 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.

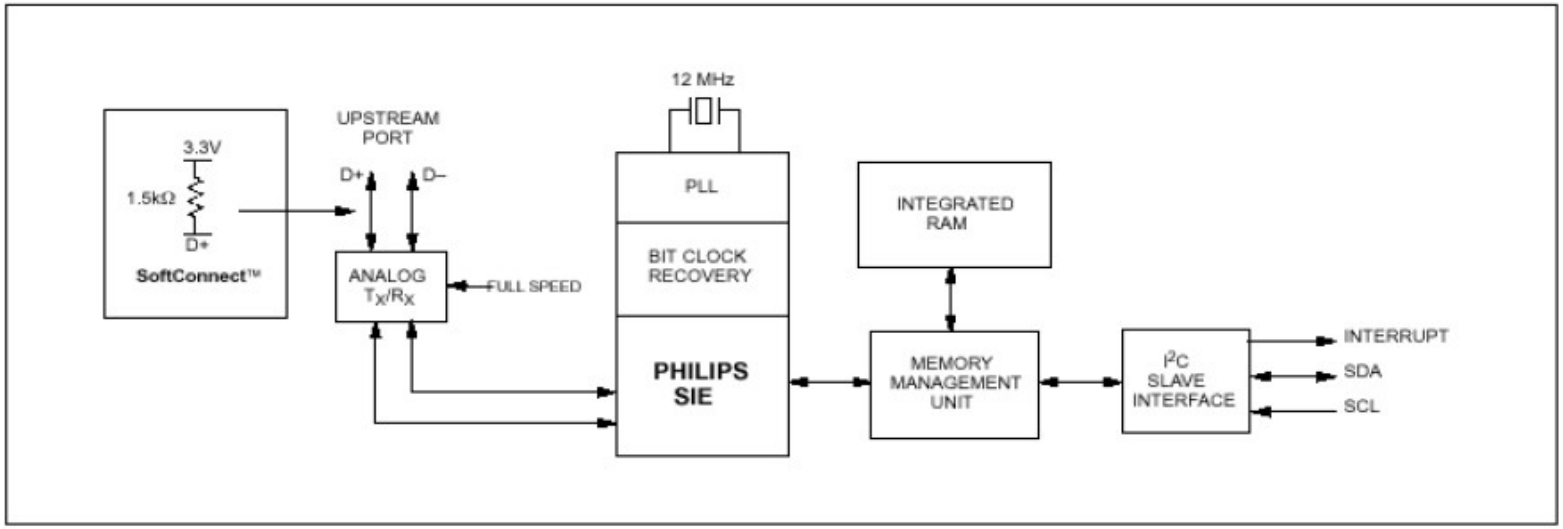
Koncentratory USB

Konwerter USB - I2C

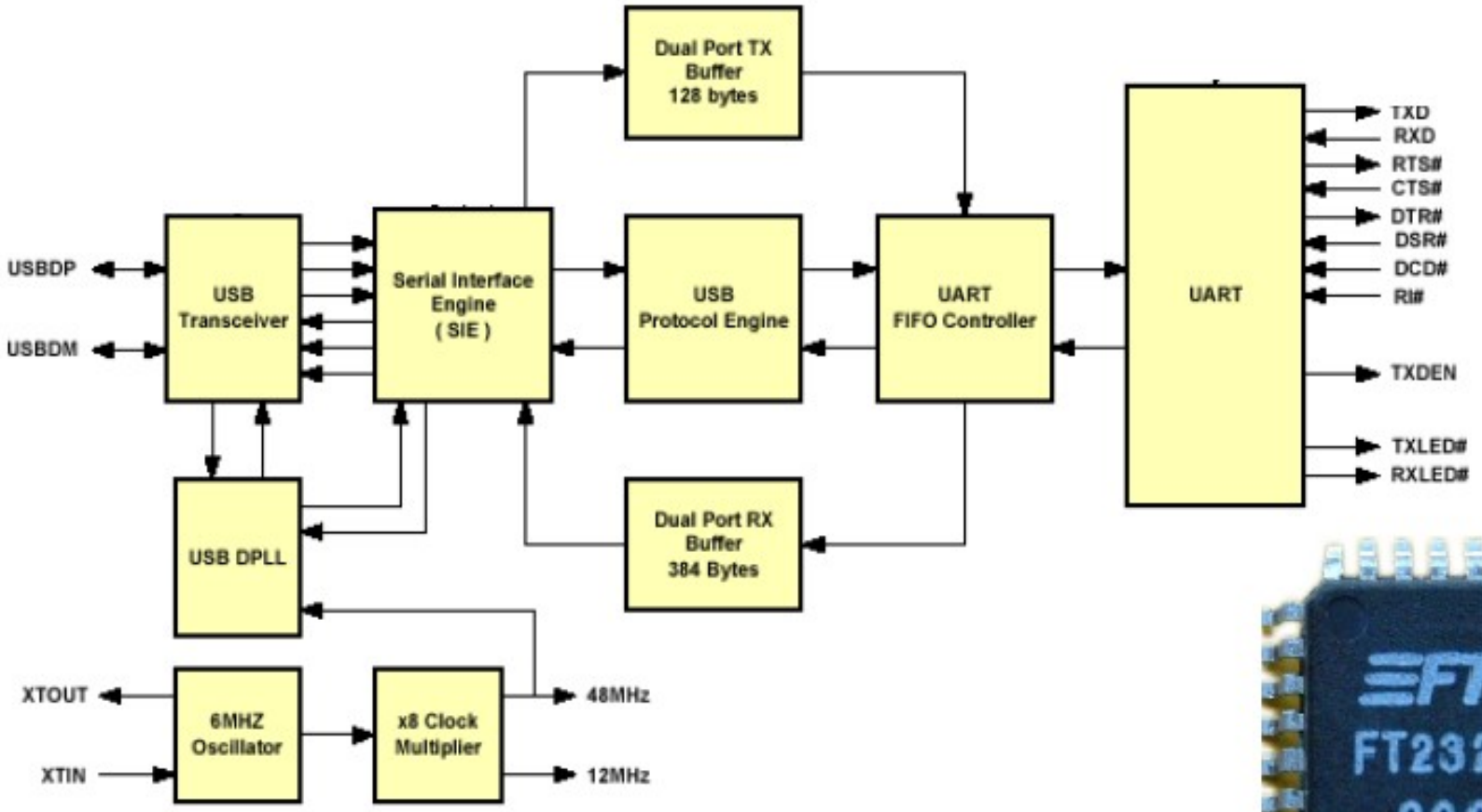
Philips PDIUSBD11 (USB to I2C)



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

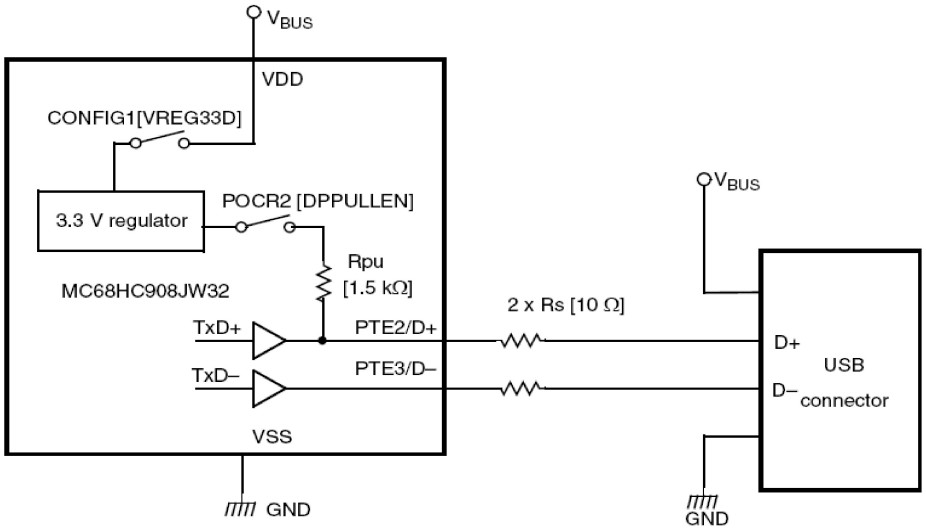
High Speed:

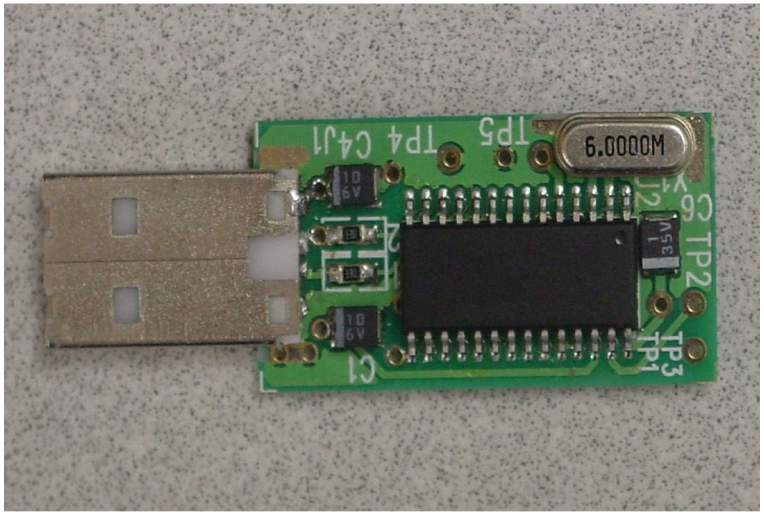
MCF 547X (72-75)	200 –266 MHz
MCF 548X (82-85)	166 – 200 MHz
MCF 537X (77-79)	240 Mhz
MCF 5253	140 MHz

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.

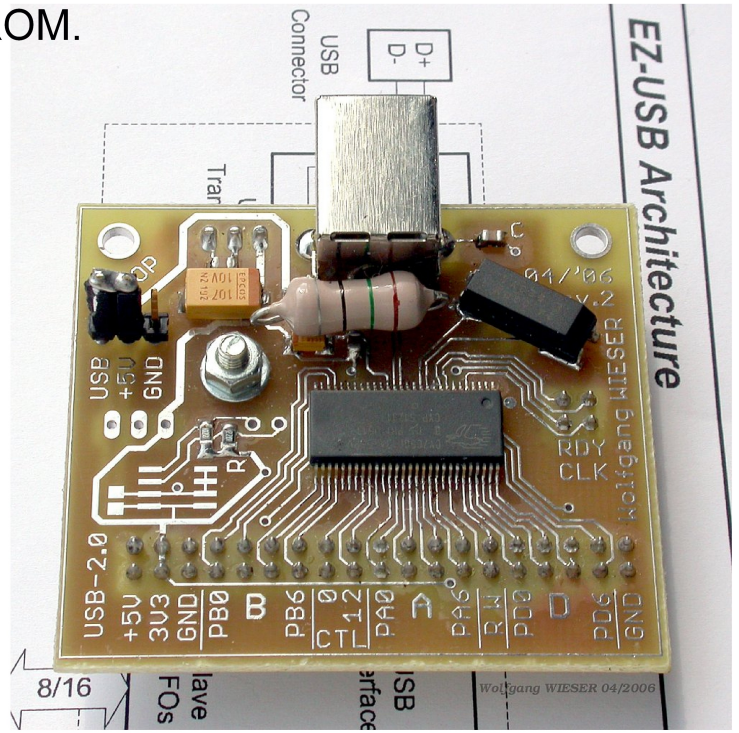




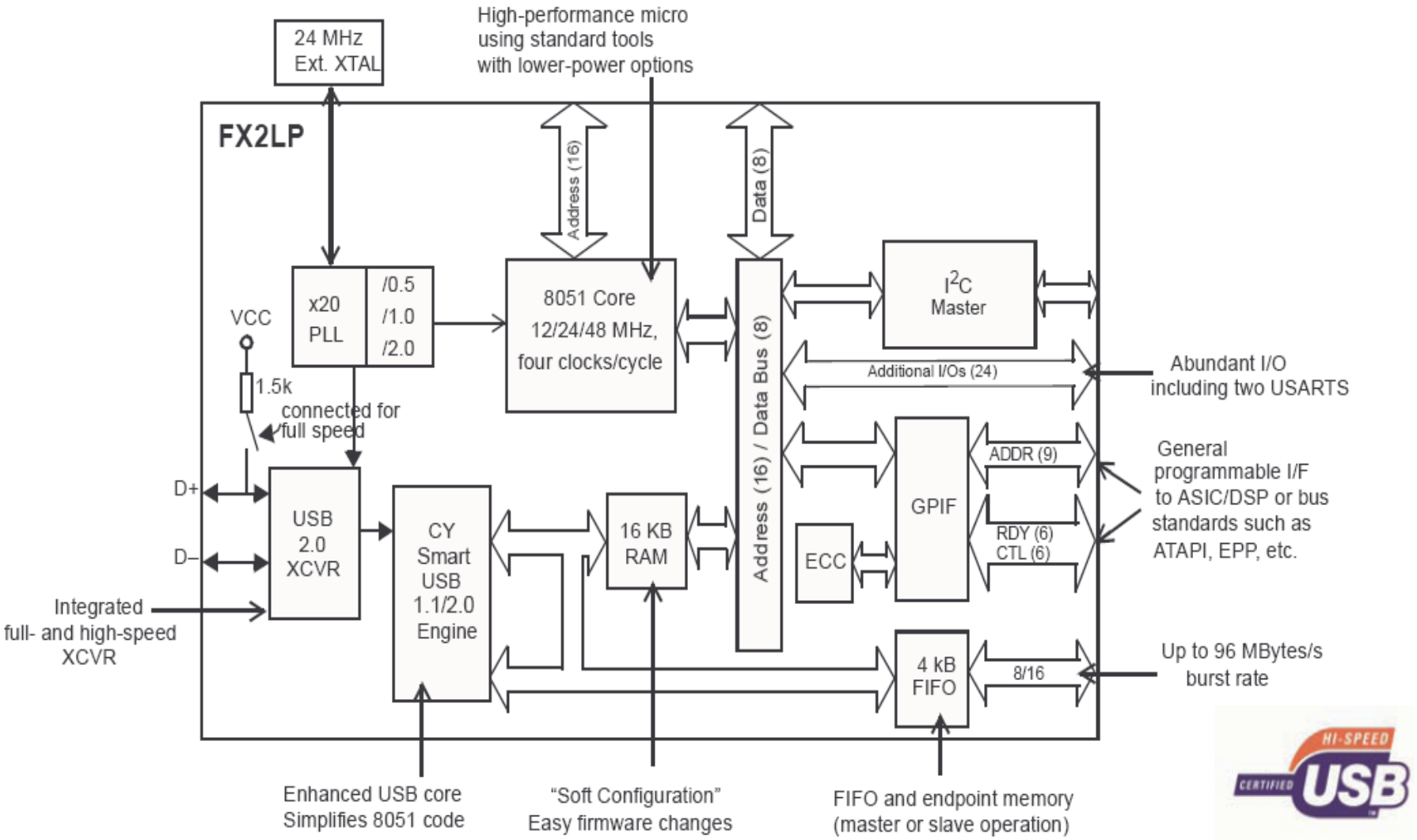
Procesor Cypress CY7C68013A

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/ISOCHRONOUS)
- Dodatkowy 64 bajtowy endpoint (BULK/INTERRUPT),
- 8- lub 16-bitowy interfejs zewnętrzny,
- Kanał DMA, GPIF (General Programmable Interface)



Procesor Cypress CY7C68013A



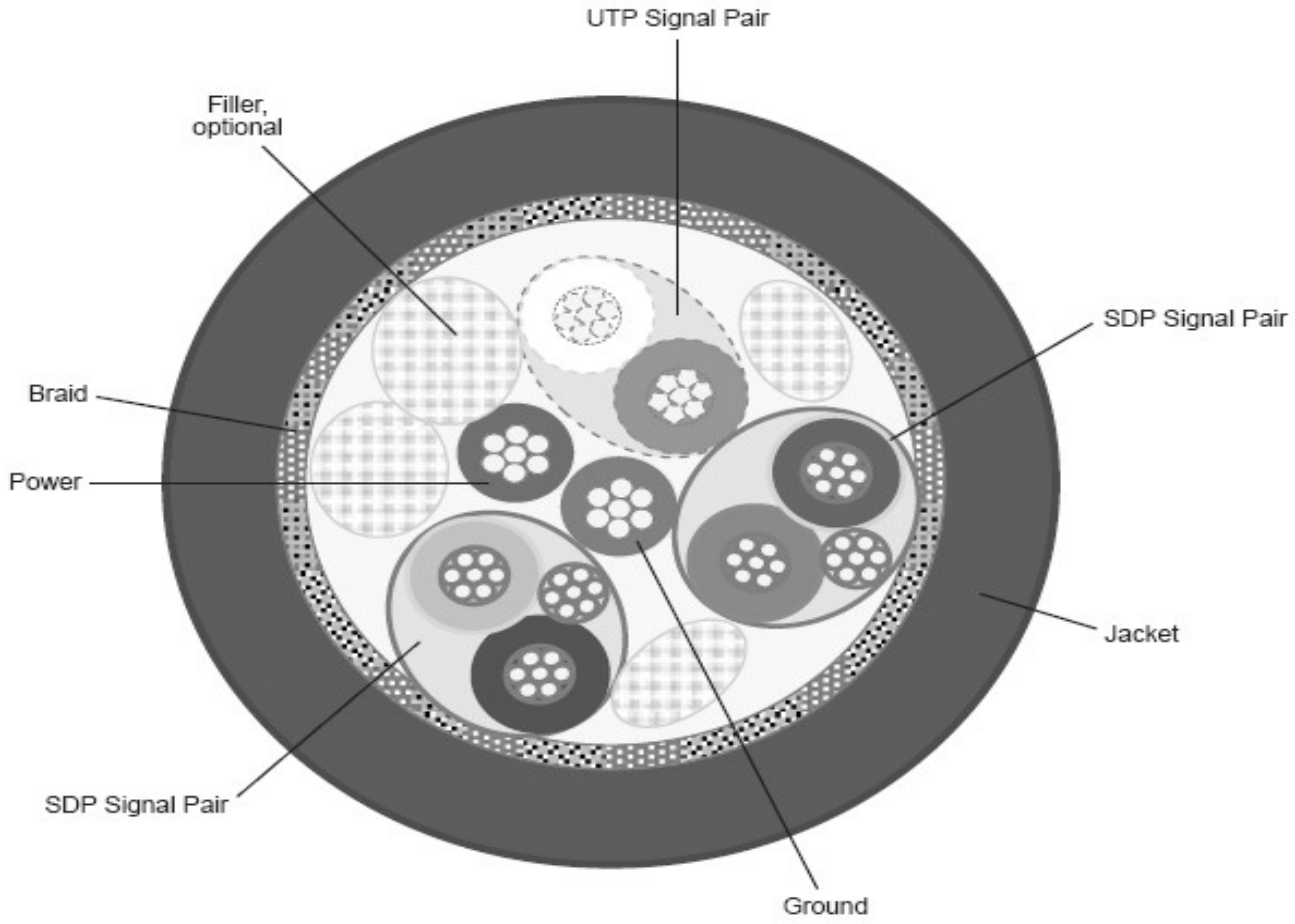
USB 3.0

SuperSpeed USB 3.0 Specification Revolutionizes An Established Standard

USB 3.0

- 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



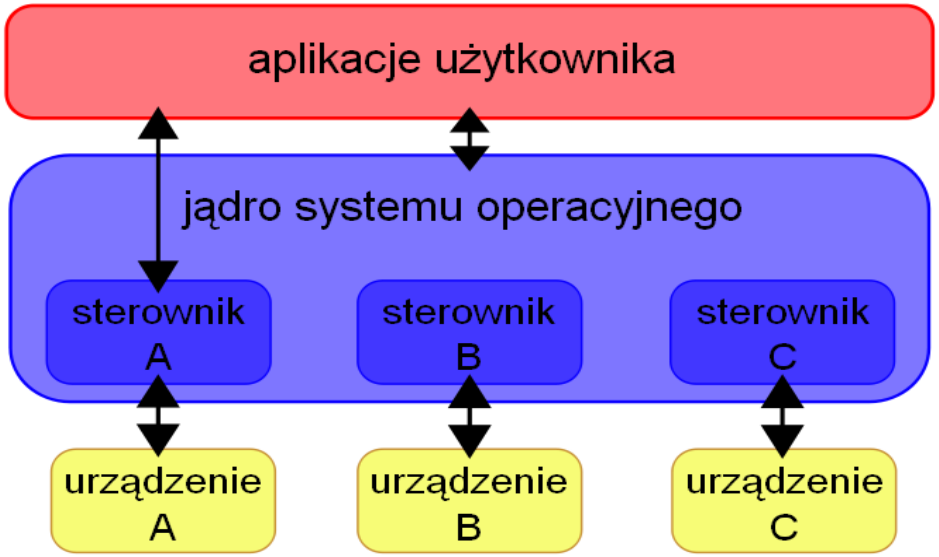
Egzamin

Termin zerowy: 19.06.2009
(godz. 10.30-11.30)

Termin 1: 24.06.2009
(godz. 10.00-11.00)

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 Programing Interface), **bezpośredni dostęp do urządzenia jest zabroniony.**

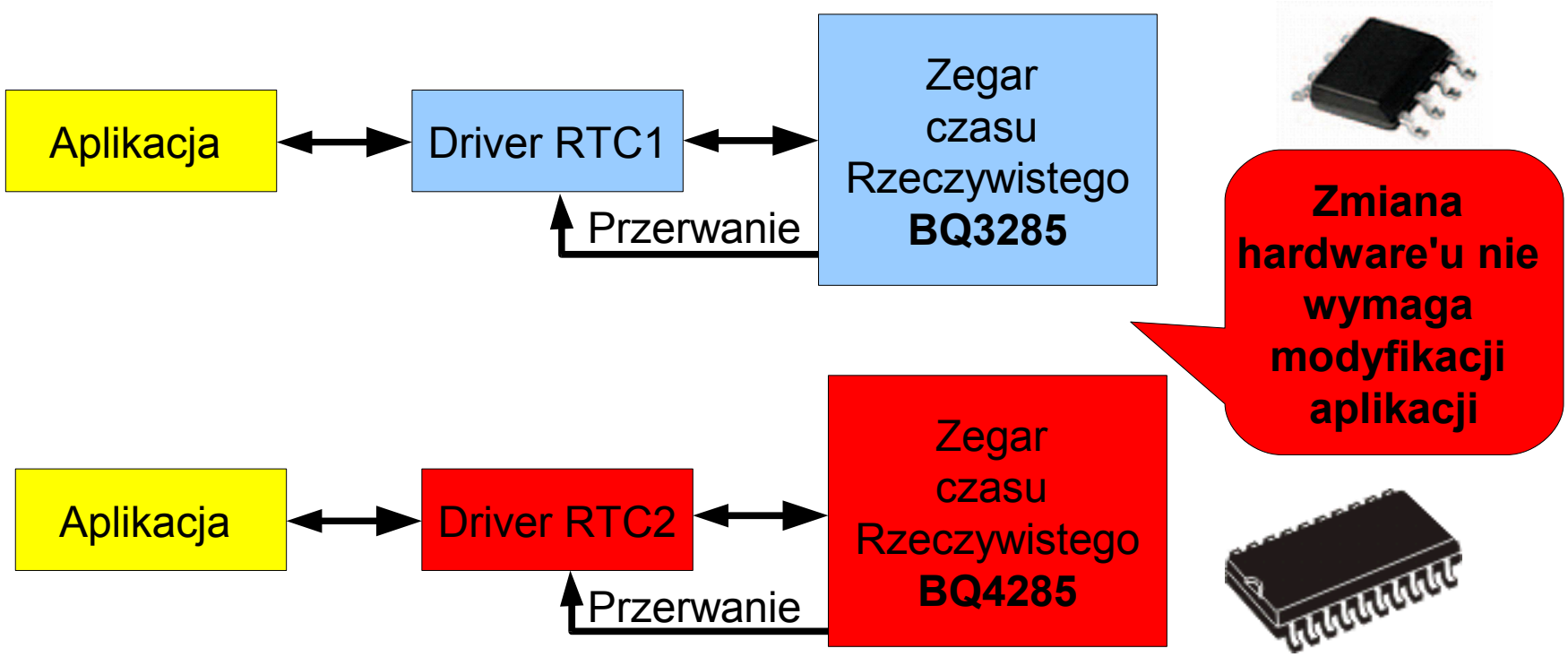
W przypadku systemów wbudowanych **aplikacje mogą się bezpośrednio odwoływać do urządzeń,** czasami trudno jest odróżnić aplikację od sterownika.

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 blokująca 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 przerwań 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 blokująca 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 informująca o zakończeniu transmisji. Wykorzystanie kanału DMA znacznie przyśpiesza 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...

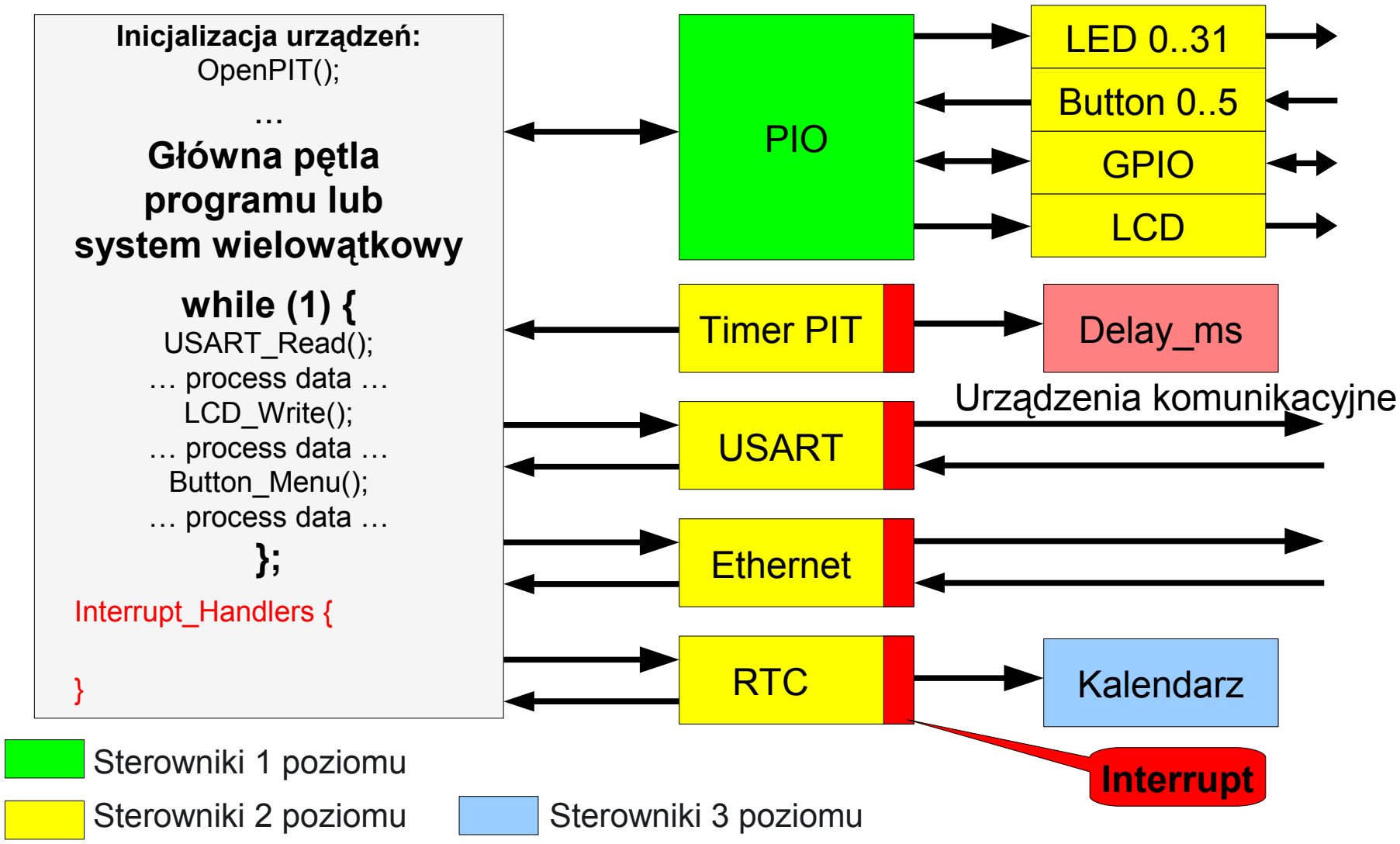
może zostać wywołana przez inną funkcję sterownika lub aplikację.

Wywołanie może nastąpić w funkcji blokującej (polling - ciągłe 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 struktury 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, unsigred int *DataToSend);

unsigned int **PIO_Status** (PIO_Struct* PoiterToPIO, unsigned int *DeviceStatus);

Funkcje pomocnicze:

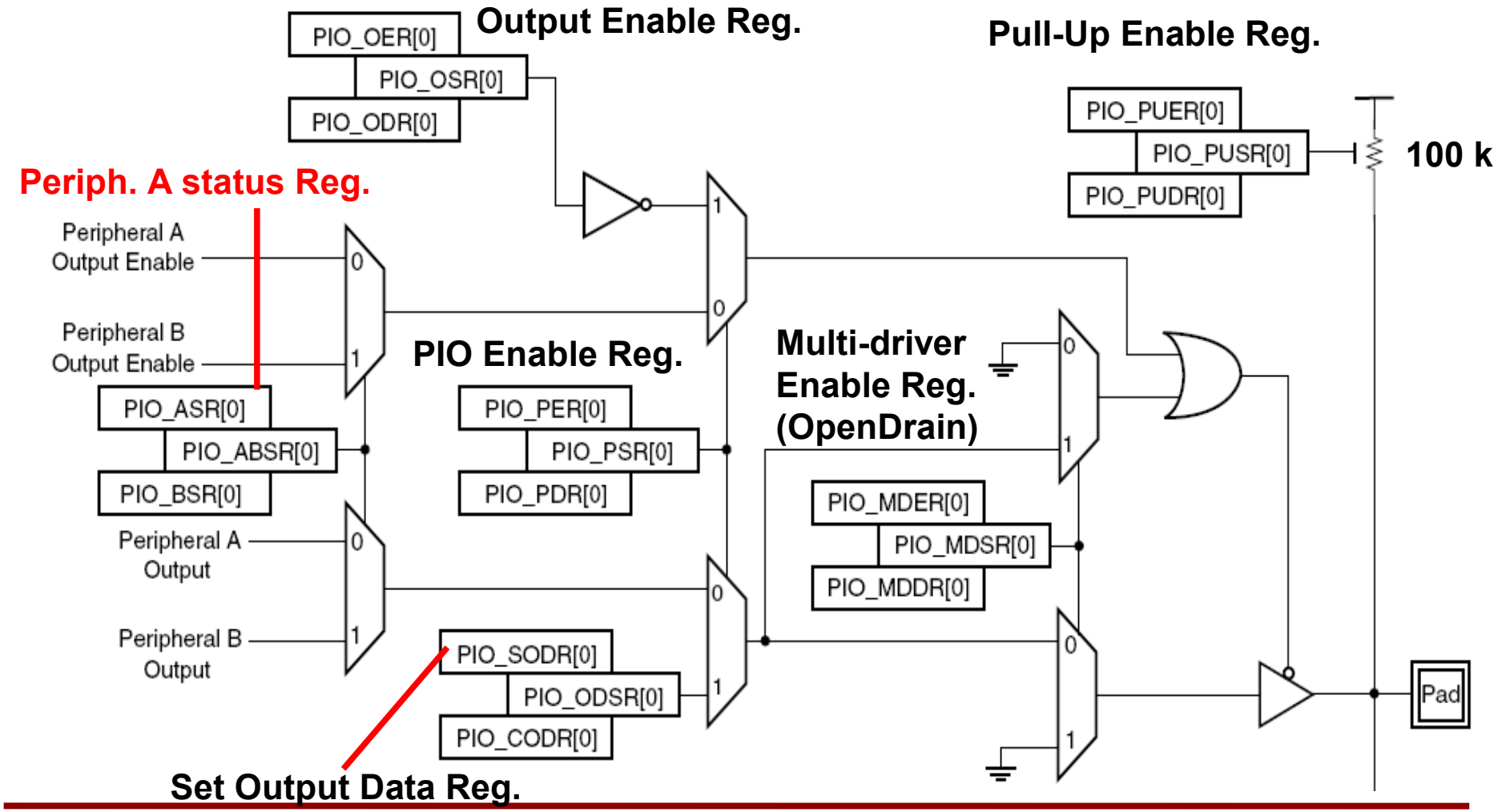
void **PIO_EnablePullUp** (unsigned int *RegistersPointer, unsigned int PortMask);

void **PIO_DisablePullUp** (unsigned int *RegistersPointer, unsigned int PortMask);

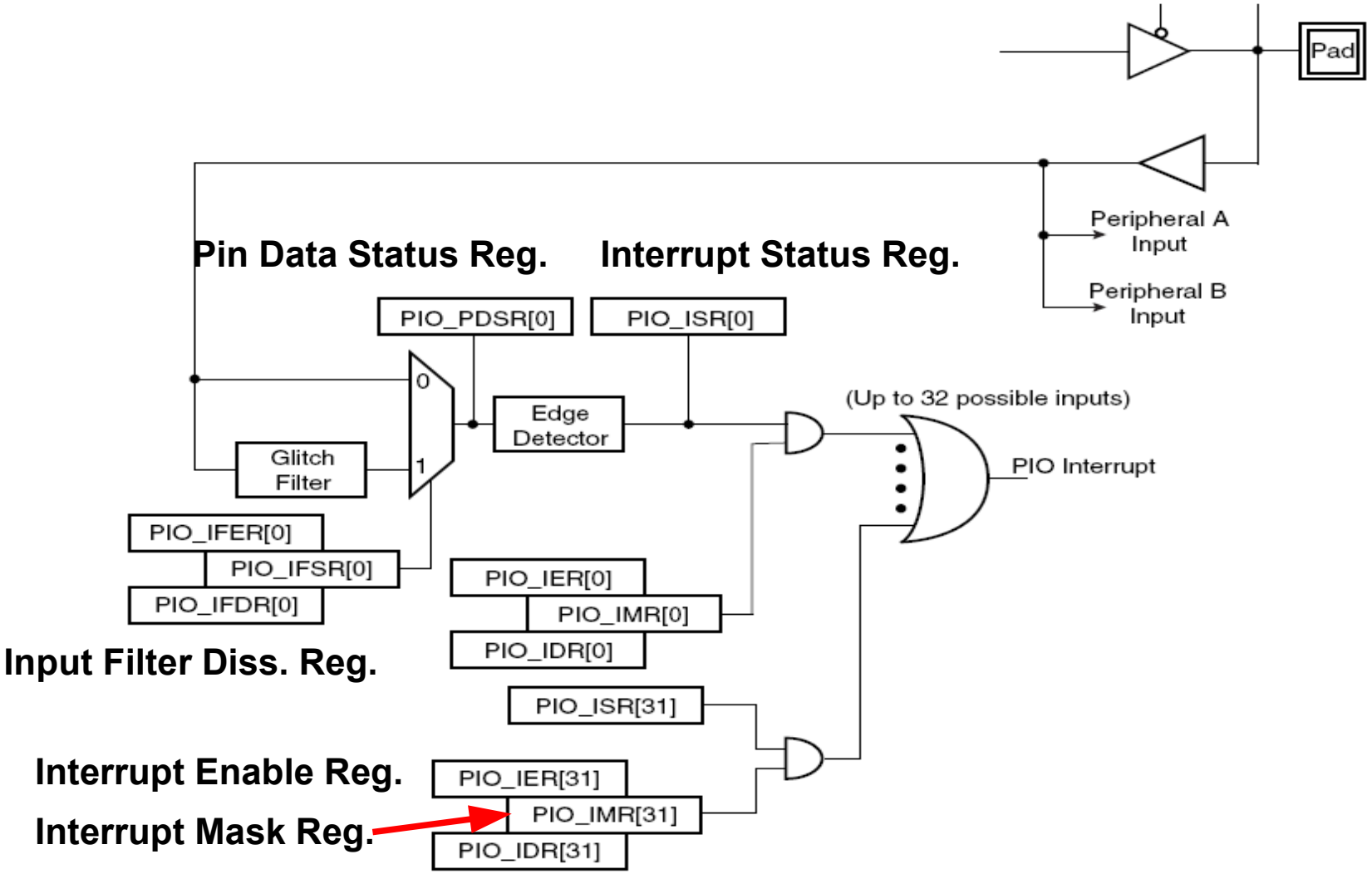
unsigned int **PIO_StatusPullUp** (unsigned int *RegistersPointer, unsigned int PortMask);

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 przerwaniami może być bardzo trudna lub nawet niemożliwa)

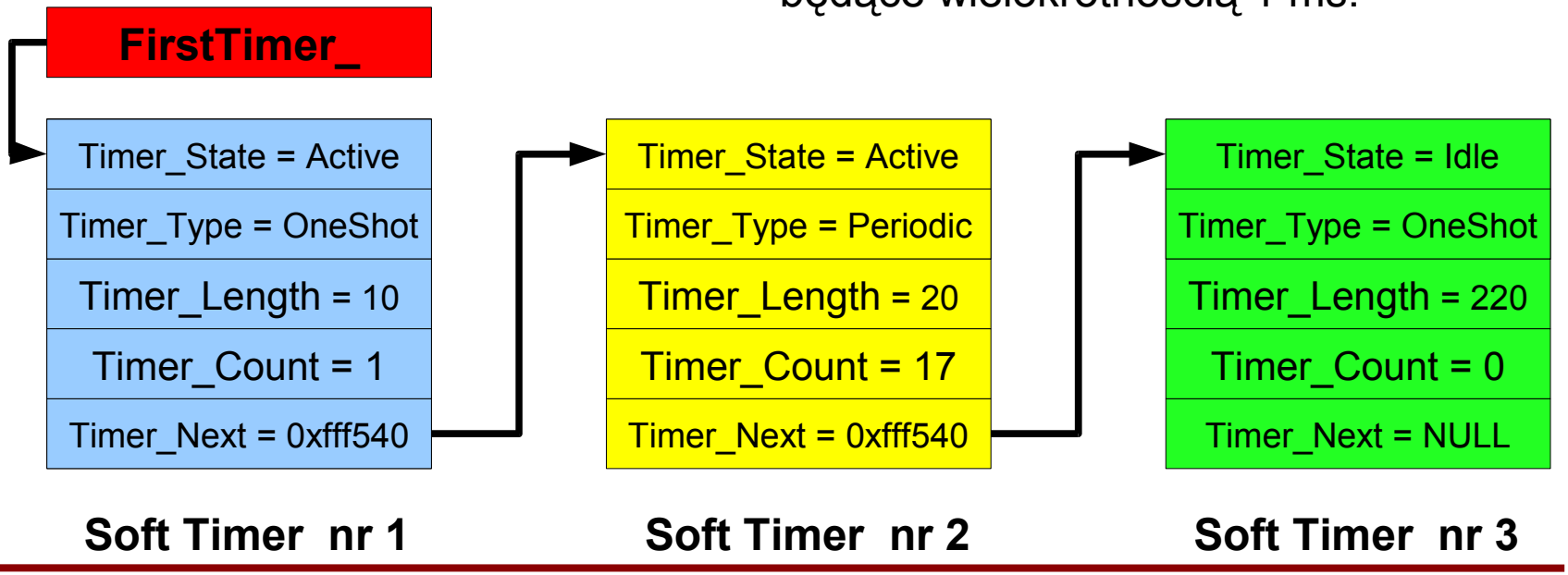
Sterownik obsługujący kilka timer'ów

```
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 expiere for each software timer */
        Timer *         Timer_next;       /* pointer to the next software timer */
}       FirstTimer, *FirstTimer_;
```

Timer sprzętowy generuje przerwanie co 1 ms.

Timery programowe mogą generować przerwania będące wielokrotnością 1 ms.

Uporządkowana lista timer'ów



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 expiere 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 miliseconds, 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 miliseconds, TimerType Type, Timer_ * TPoin){
  if (Tpoin->Timer_State != Idle)
        return -1;
  Tpoin->Timer_State = Active;
  Tpoin->Timer_Type = Type;
  Tpoin->Timer_Length = miliseconds / 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 wygenerownie 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 miliseconds, TimerType = OneShot);
                int Wait_For ();
                void Cancel ();

                TimerState      State;
                TimerType       Type;
                unsigned int    Length;


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

Plik startowy
(startup file)

Struktura pliki startowego

Program startowy uruchamiany jest zaraz po **sygnale resetu** w celu konfiguracji podstawowych zasobów procesora.

Plik startowy jest zwykle napisany w języku asemblera ze względu na odwołania do specyficznych zasobów procesora (dostępnych z poziomu asemblera) **i uruchamiany przed** programem napisanym w języku wyższego poziomu:

- Alokacja pamięci na stosy dla poszczególnych trybów pracy procesora, inicjalizacja wskaźników dla stosów,
- Konfiguracja pamięci (SRAM, przemapowanie pamięci FLASH, wyczyszczenie pamięci),
- Iniclalizacja tablicy wektorów wyjątków,
- Skopiowanie kodu systemu operacyjnego lub aplikacji do pamięci RAM,
- Inicjalizacja zmiennych globalnych w pamięci RAM (skopiowanie danych, wyzerownie, przypisanie wartości),
- Konfiguracja wymaganych urządzeń peryferyjnych,
- Inicjalizacja systemu przerwań,
- Zmiana trybu pracy procesora (jeżeli wymagane),
- Wywołanie funkcji main().

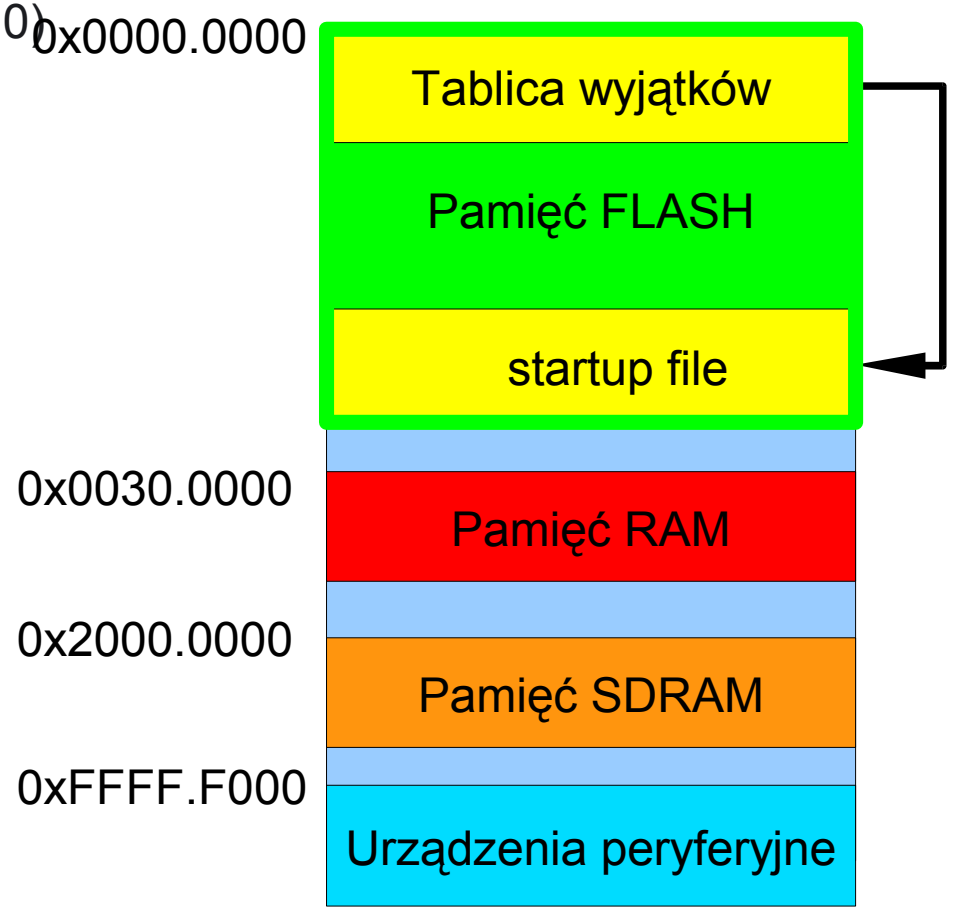
Struktura pliki startowego

Wektor RESET (pod adresem 0)

```
.section .text
reset_handler:
ldr     pc, =_low_level_init
/*       Inicjalizacja...          */
_low_level_init:


_stack_init:
_init_data:
_init_bss:


_branch_main:
```



Struktura pliki startowego

Program w funkcji main powinien pracować w nieskonczonej pętli, nie może zostać wykonany rozkaz return albo exit.

```
...
...
_branch_main:
        ldr      r0, =main
        mov      lr, pc
        bx       r0
...
...
void main (void) {
                While (1)
                {

                        program główny

                }
        return 0;
}
```

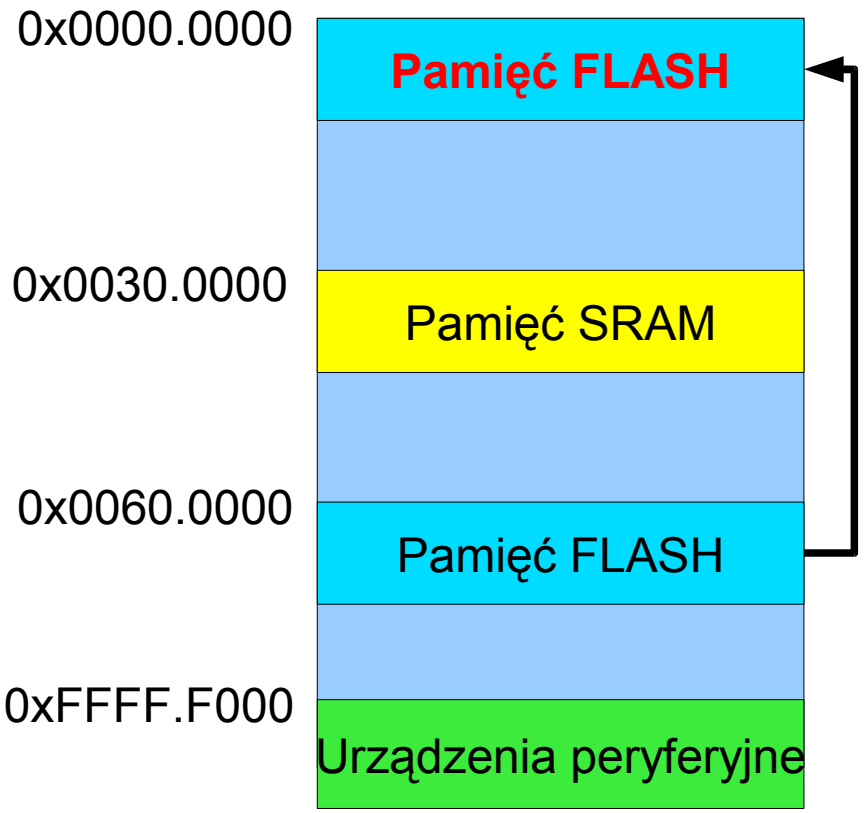
Struktura pliki startowego

Konfiguracja urządzeń krytycznych, niezbędnych do pracy procesora:

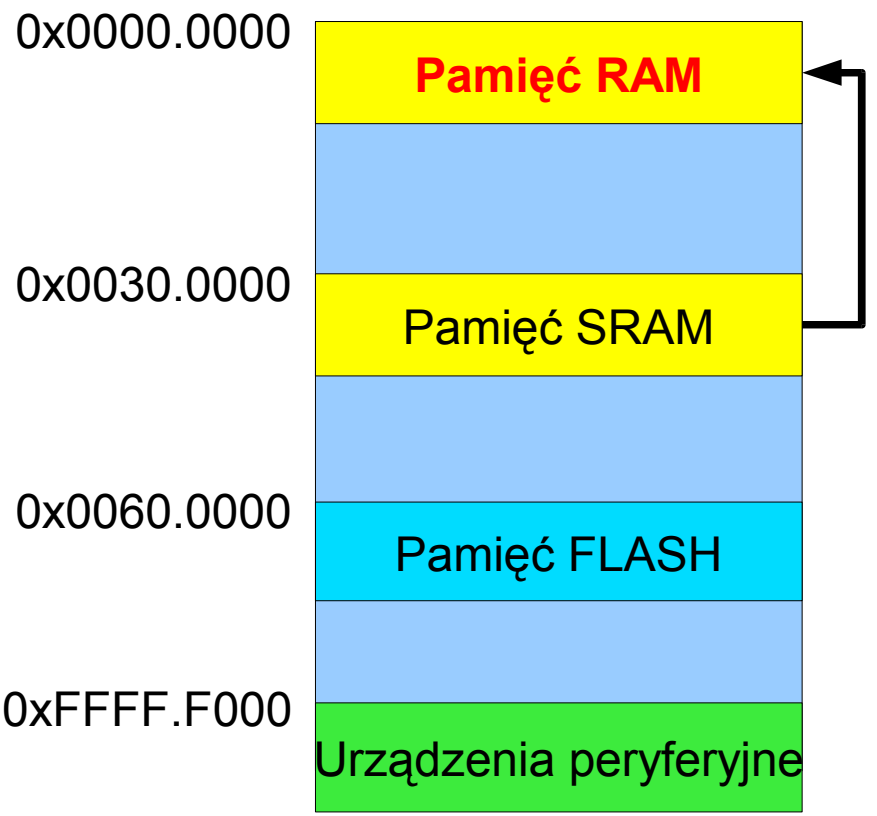
- Konfiguracja modułu dostarczającego sygnał zegarowy (PLL). Po resecie procesor pracuje z tzw. wolnych zegarem (wewnętrzny generator RC),
- Konfiguracja modułu sterującego pamięcią FLASH, RAM (liczba cykli opóźnienia pamięci – WaitStates),
- Przemapowanie pamięci FLASH-SRAM,
- Konfiguracja licznika Watch-Dog (po resecie Watch-Dog jest włączony),
- Konfiguracja modułu AIC (przypisanie domyślnych handlerów do przerwań),
- Inicjalizacja wskaźników stosów dla poszczególnych trybów pracy (Mode, IRQ, FIQ,...),
- Odblokowanie wejścia NRST sygnału reset

Podmiana pamięci po uruchomieniu

Mapa pamięci podczas uruchamiania



Mapa pamięci po przemapowaniu

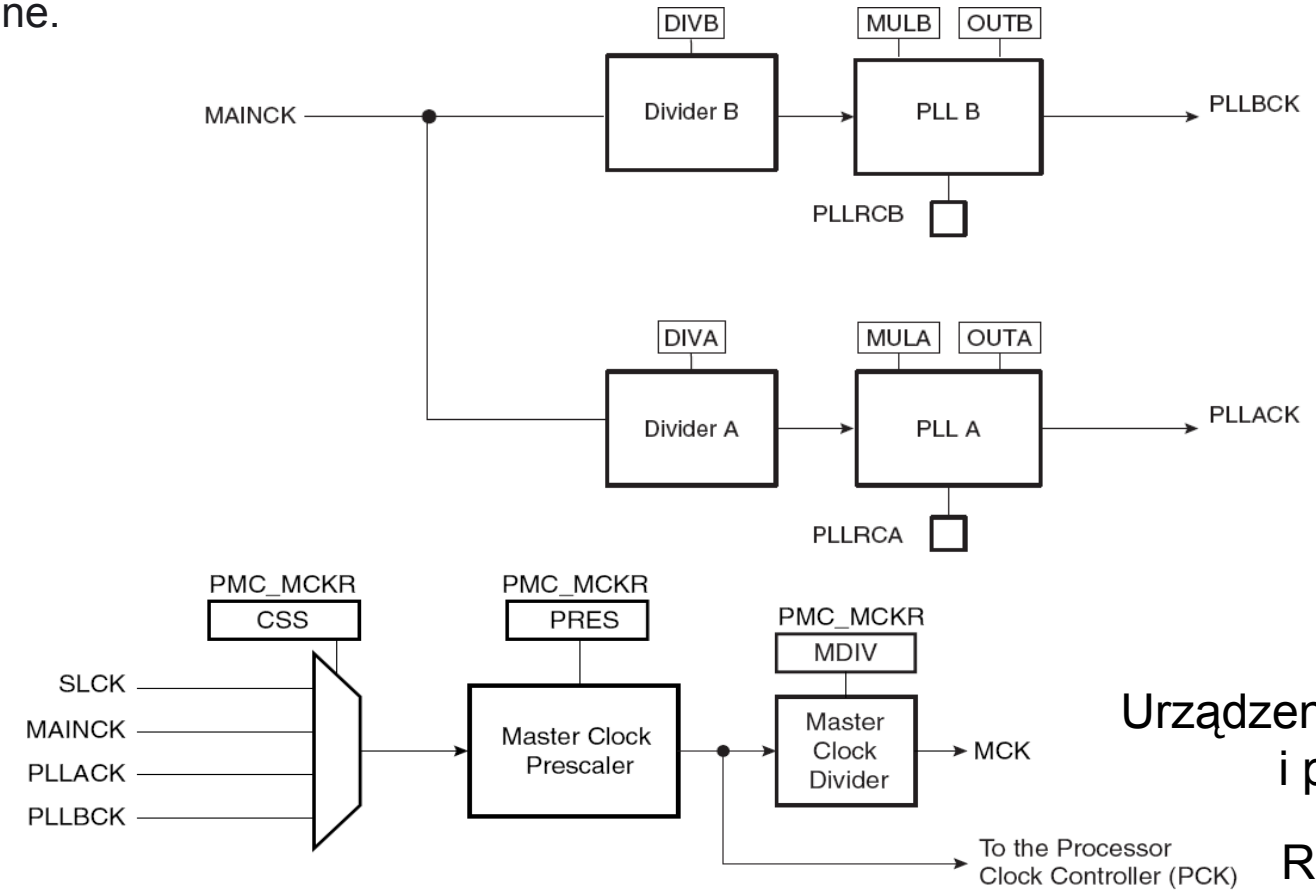


Podmiana pamięci FLASH następuje po wykonaniu programu startup (rejestr REMAP, najmłodszy bit)

Konfiguracja i wybór sygnału zegarowego - rozdział 28 (1)

Po restarcie procesor pracuje z tzw. wolnym zegarem (SLOWCLK) o częstotliwości f = 32768 Hz. Zegar ten jest zawsze dostępny, generowany jest przez wbudowany generator RC.

Po restarcie generator kwarcowy oraz blok pętli synchronizacji fazowej PLL (Phase Locked Loop) są wyłączone.



Urządzenia peryferyjne i pamięci

Rdzeń ARM

Generator z pętlą PLL (1)

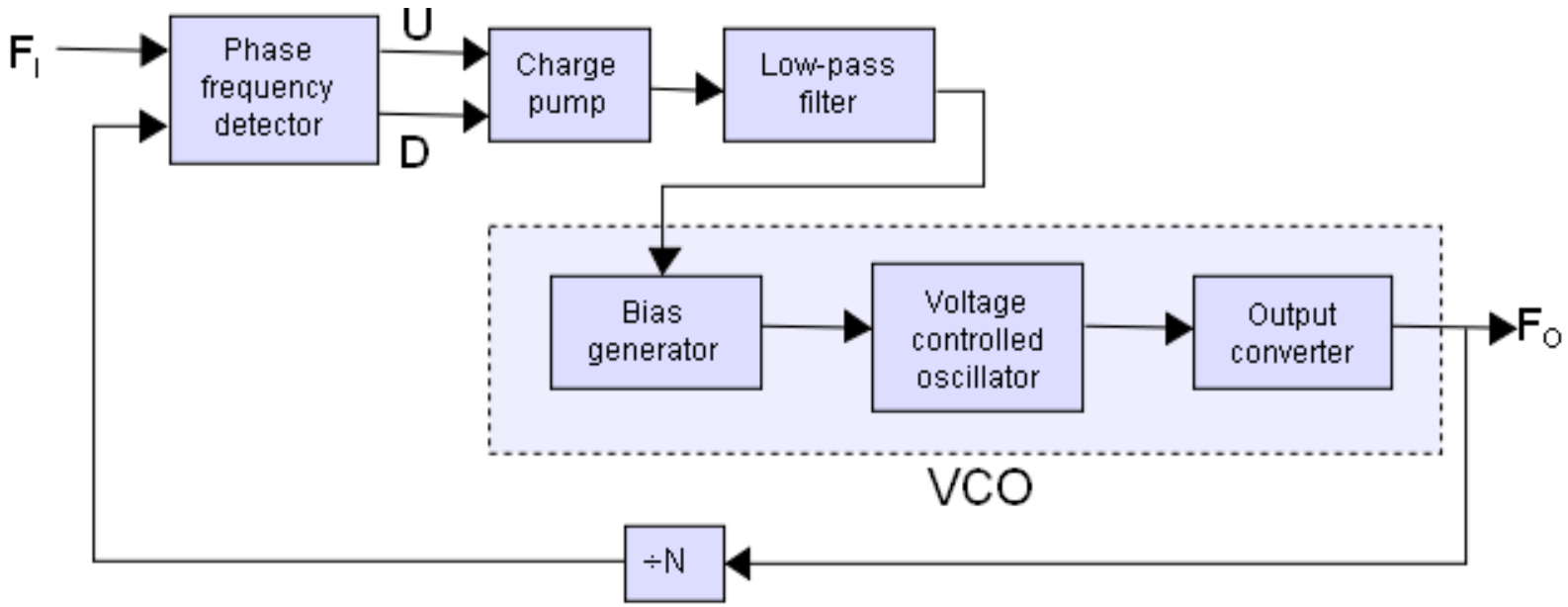
Pętla synchronizacji fazy, pętla sprzężenia fazowego, PLL (ang. Phase Locked Loop) - układ elektroniczny działający na zasadzie sprzężenia zwrotnego, służący do automatycznej regulacji częstotliwości.

Stosowana w syntezerach częstotliwości heterodyny w odbiornikach radiowych, telewizyjnych oraz w **generatorach częstotliwości wzorcowych i powielaczach częstotliwości.**

Generator PLL zbudowany jest z:

- generatora sygnału referencyjnego (rezonatora kwarcowego),
- detektora fazy,
- filtru dolnoprzepustowego,
- generatora przestrajanego napięciem - VCO,
- pętli sprzężenia zwrotnego, w której występuje dzielnik częstotliwości.

Generator z pętlą PLL (2)



Sygnał wysokiej częstotliwości generowany przez VCO jest sygnałem na wyjściu całego urządzenia (Fo). Jednocześnie podawany jest do pętli sprzężenia zwrotnego, w której zwykle następuje dzielenie jego częstotliwości tak, aby była równa częstotliwości sygnału referencyjnego (Fi). Dzięki temu różnica faz obu sygnałów - uzyskana w detektorze fazy - po przejściu przez filtr steruje generatorem VCO (generator sterowany napięciem, napięci rośnie => częstotliwość rośnie), korygując jego częstotliwość.

Konfiguracja i wybór sygnału zegarowego (2)

Procedura włączenia generatora z pętlą PLL:

1. Włączenie generatora kwarcowego. Po włączeniu należy odczekać, aż częstotliwość się ustabilizuje (bit PMC_MOSCS).
2. Konfiguracja pętli PLLA, f = 16 367 660*110/9 = ~200 MHz. Po włączeniu należy odczekać, aż PLLA się zablokuje (bit PMC_LOCKA), a częstotliwość ustabilizuje (bit PMC_MCKRDY).
3. Konfiguracja wyprowadzenia taktującego procesor na PLLA (w przykładzie dodatkowy podzielnik równy 2), bit AT91C_PMC_CSS_PLLA_CLK. Odczekanie, aż czestotliwość się ustabilizuje.

Konfiguracja PLLA:

```
AT91C_BASE_PMC->
                    PMC_PLLAR = AT91C_CKGR_SRCA    |    /* programming PLL */
                    AT91C_CKGR_OUTA_2              |    /* parametry elektryczne  */
                    (0x3F << 8)                    |    /* counter = 63 */
                    (AT91C_CKGR_MULA & (0x6D << 16)) |  /* mnożnik 109 */
                    (AT91C_CKGR_DIVA & 9);              /* dzielnik 9 */
```

Fref = 16 367 660 Hz

fout = fref * (MULA+1) / DIVA = 16 MHz * 110 / 9 => 200 MHz

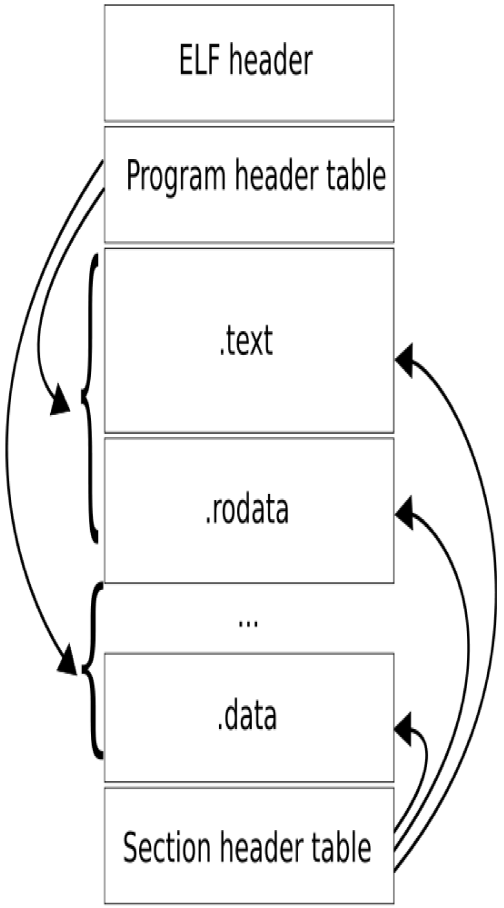
f_{MCK} = fout / 2 => ~100 MHz

Analiza pliku lowlevel.c

COFF vs ELF

- **COFF (Common Object File Format) –** standard plików wykonywalnych, relokowalnych i bibliotek dynamicznych opracowany na potrzeby systemów operacyjnych bazujących na systemie Unix. COFF miał zastąpić standard plików **a.out**. Wykorzystywany na różnych systemach, również Windows. Obecnie standard COFF wypierany jest przez pliki ELF.

- **ELF (Executable and Linkable Format) –** standard plików wykonywalnych, relokowalnych, bibliotek dynamicznych i zrzutów pamięci wykorzystywany na różnych komputerach i systemach operacyjnych, np.: rodzina x86, PowerPC, OpenVMS, BeOS, konsole PlayStation Portable, PlayStation 2, PlayStation 3, Wii, Nintendo DS, GP2X, AmigaOS 4 oraz Symbian OS v9.
- Przydatne narzędzia:
  - readelf
  - elfdump
  - objdump



Źródło: wikipedia

Skrypt linkera

```
/* elf32-littlearm.lds 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: $(OBJS)
$(LD) $(LDFLAGS) -Ttext 0x20000000 -Tdata 0x300000 -n -o $(OUTFILE_SDRAM).elf $
(OBJS)
```

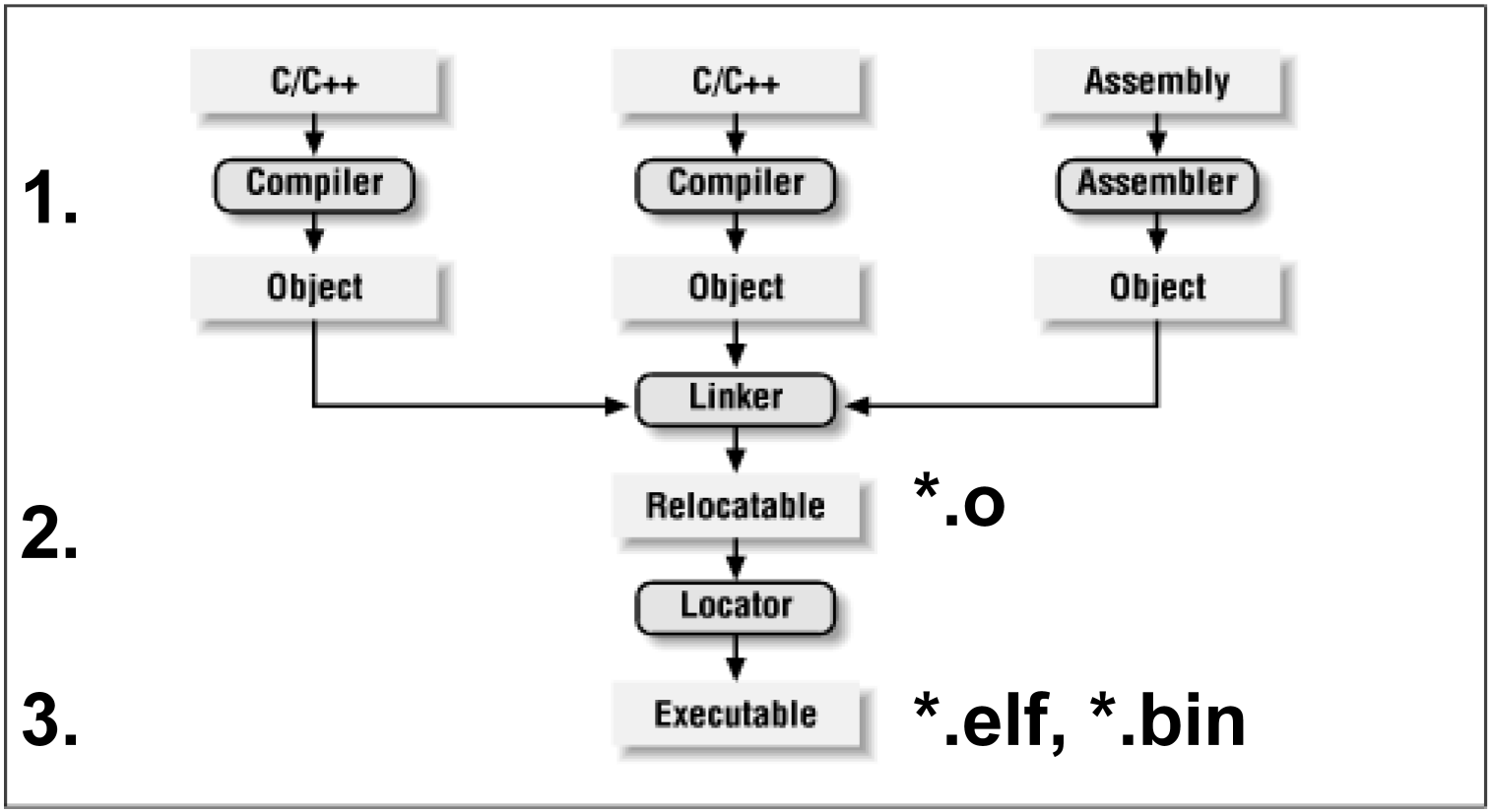
Skrypt linkera dla procesora ARM AT91SAM9263

```
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 -Wl,--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)
```

Proces kompilacji programu



- 1 faza – kompilacja plików źródłowych → pliki binarne, relokowalne
- 2 faza – linkowanie plików relokowalnych → plik binarny, relokowalny
- 3 faza – generowanie pliku wykonywalnego (przypisanie adresów)

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 adresewanie urządzeń przy użyciu 8-bitowego adresu,
- Umożliwia adresewanie 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 opóźnia generowanego przez timer,
- Jaką wartość należy wpisac do rejestru DIV, aby timer odliczył czas równych 1 ms?
- Jaka jest wartość błędu popełnionego podczas odmierzania 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 rejestów, 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;
}
```