KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

”Procesory ARM w systemach wbudowanych”
”Programowanie procesorów ARM”

Prezentacja jest współfinansowana przez
Unię Europejską w ramach
Europejskiego Funduszu Społecznego w projekcie pt.

„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”

Prezentacja dystrybuowana jest bezpłatnie

Dariusz Makowski

Katedra Mikroelektroniki i Technik Informatycznych

tel. 631 2720

dmakow@dmcs.pl

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

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Zakres przedmiotu

- Systemy mikroprocesorowe, systemy wbudowane
- Laboratorium
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Interfejsy w systemach wbudowanych
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Interfejsy w systemach wbudowanych

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

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),
  - Interfajs Two-wire Interface (TWI),
  - Interfejs Synchronous Serial Controller (SSC)
  - Interfejs I2C,
  - Interfejs Controlled Area Network (CAN),
  - Interfejs Universal Serial Bus (USB),
  - Interfejs Ethernet 10/100.

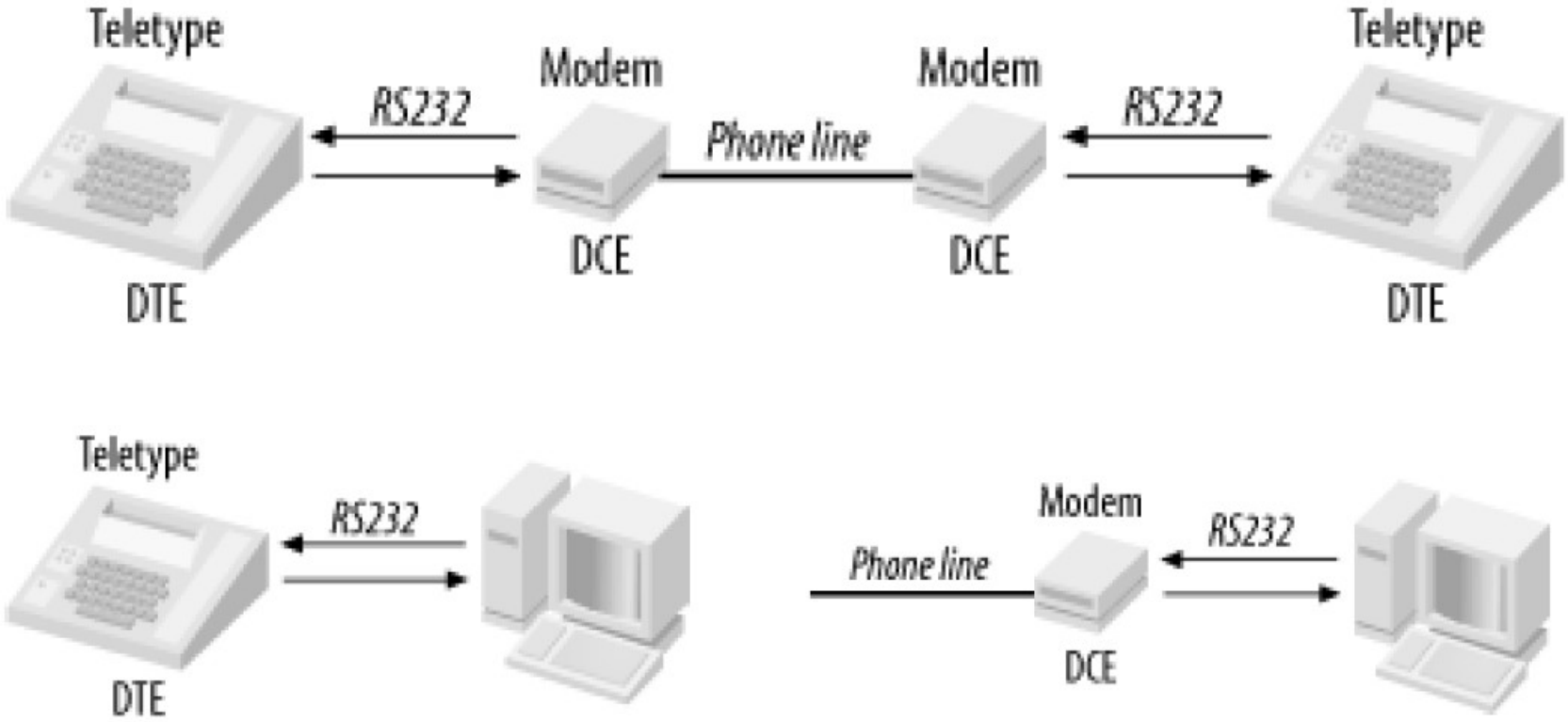
Moduł transceivera szeregowego UART (Universal Asynchronous Receiver/Transmitter module)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

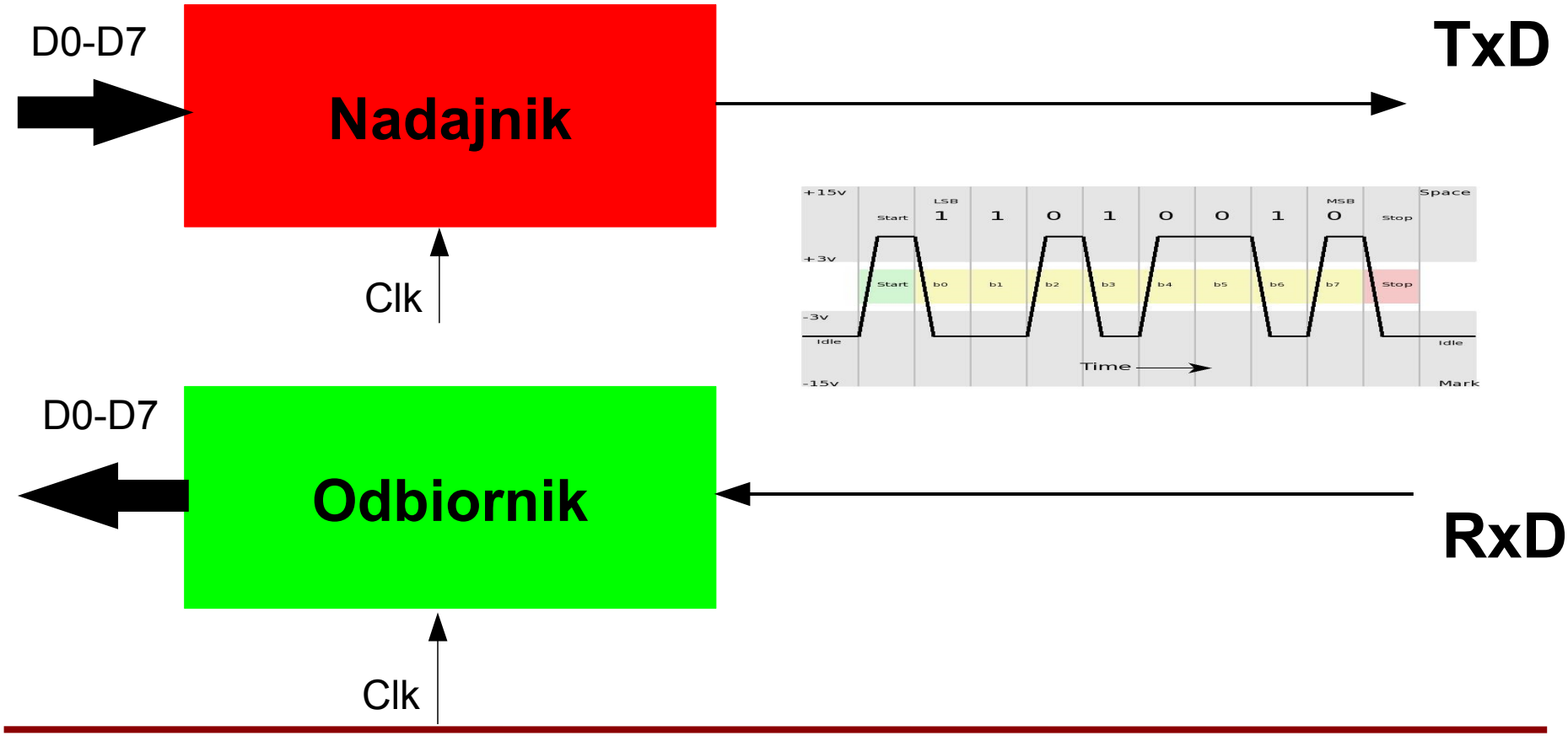
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Interfejs szeregowy EIA RS232



Transceiver UART

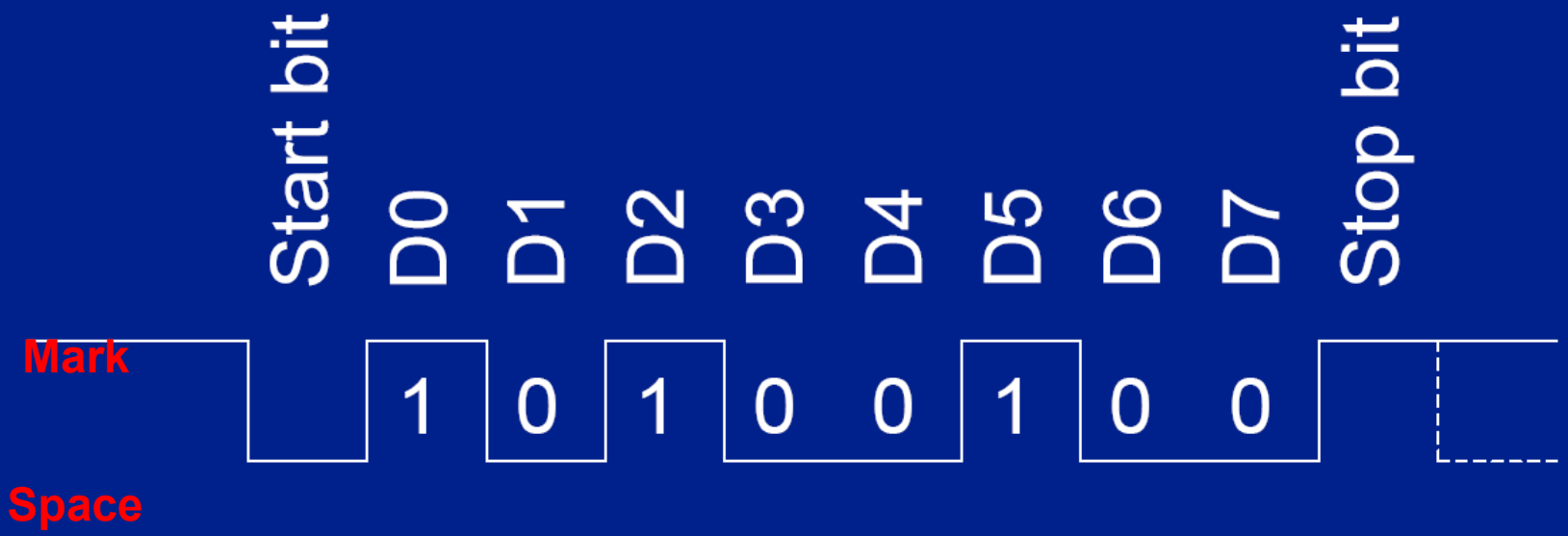
Rejestr przesuwny



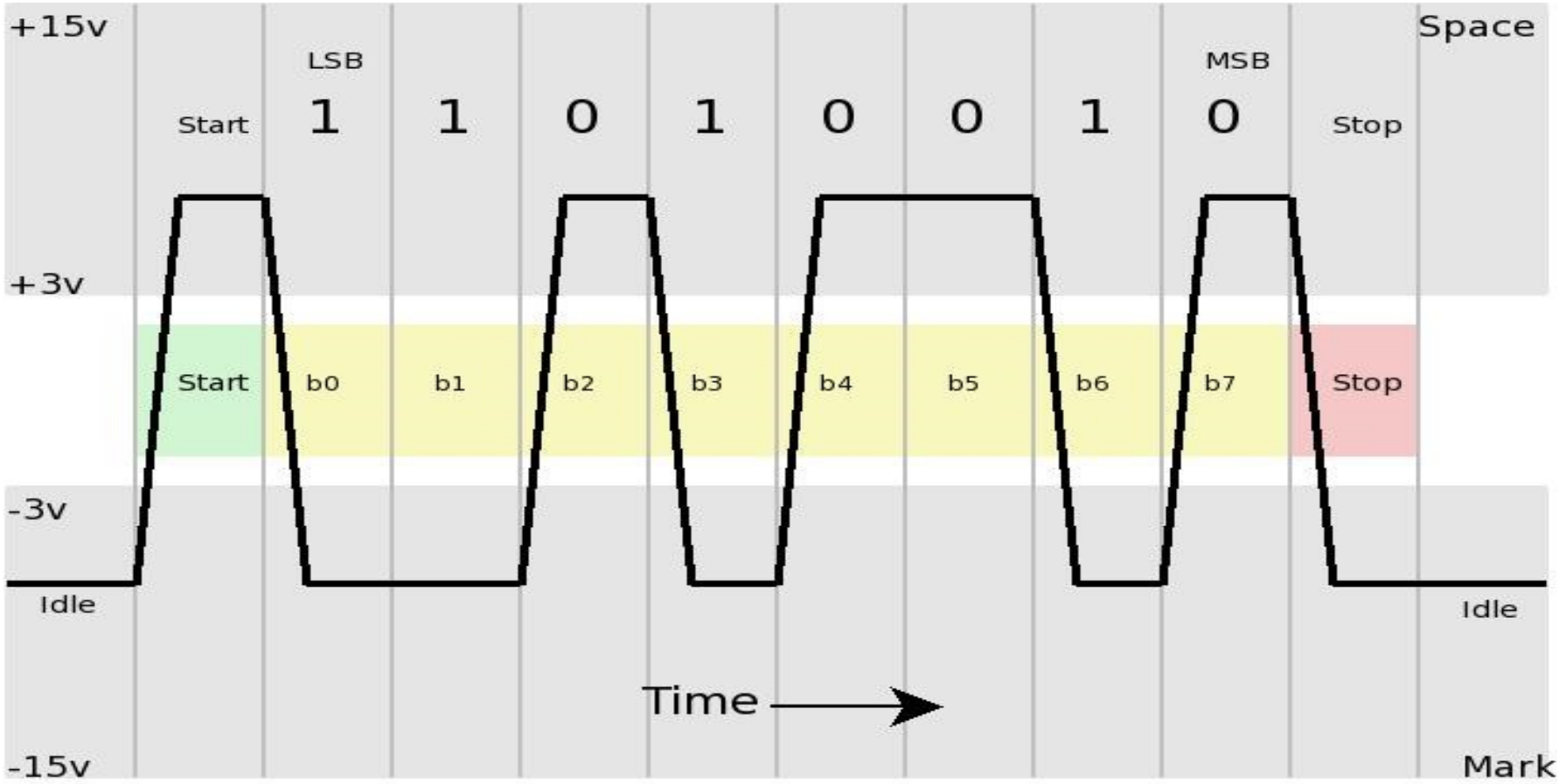
Ramka danych transmitera UART (1)

Asynchronous 8 bit waveform example

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

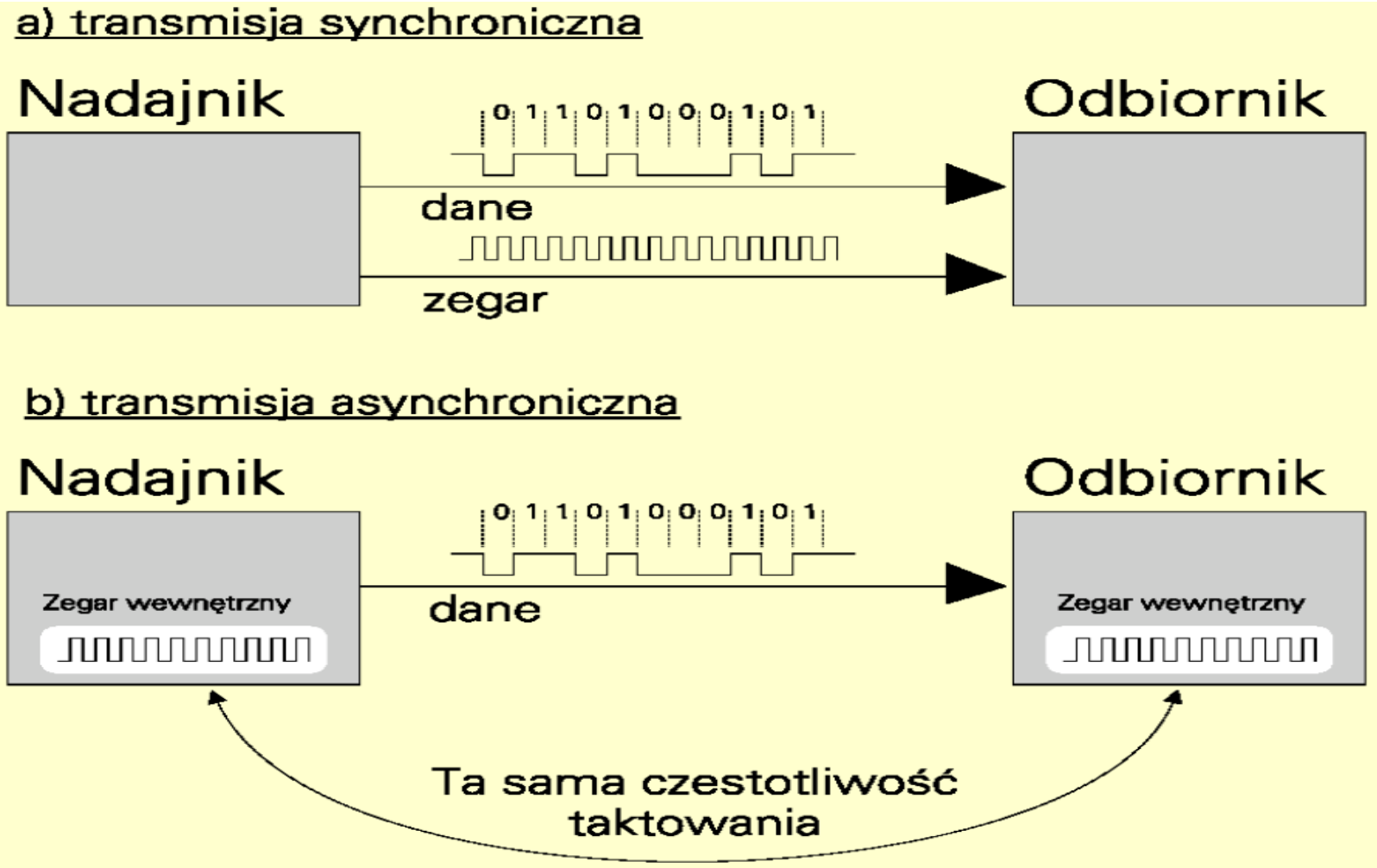


Ramka danych transmitera UART (2)



Przesylana dana: 0100.1011b = 0x4B

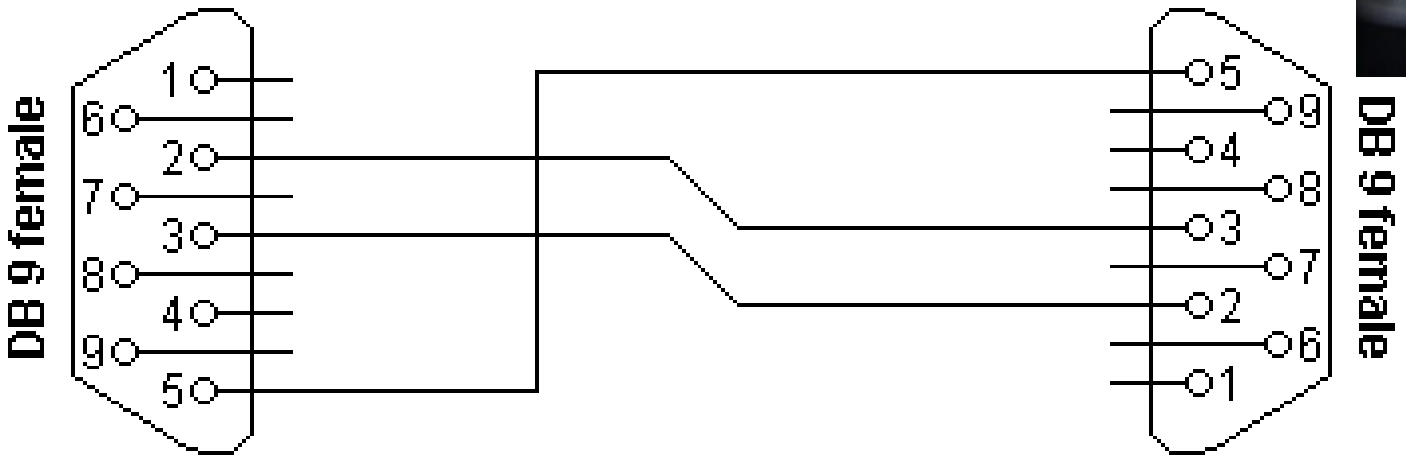
Transmisja synchroniczna vs asynchroniczna?



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)

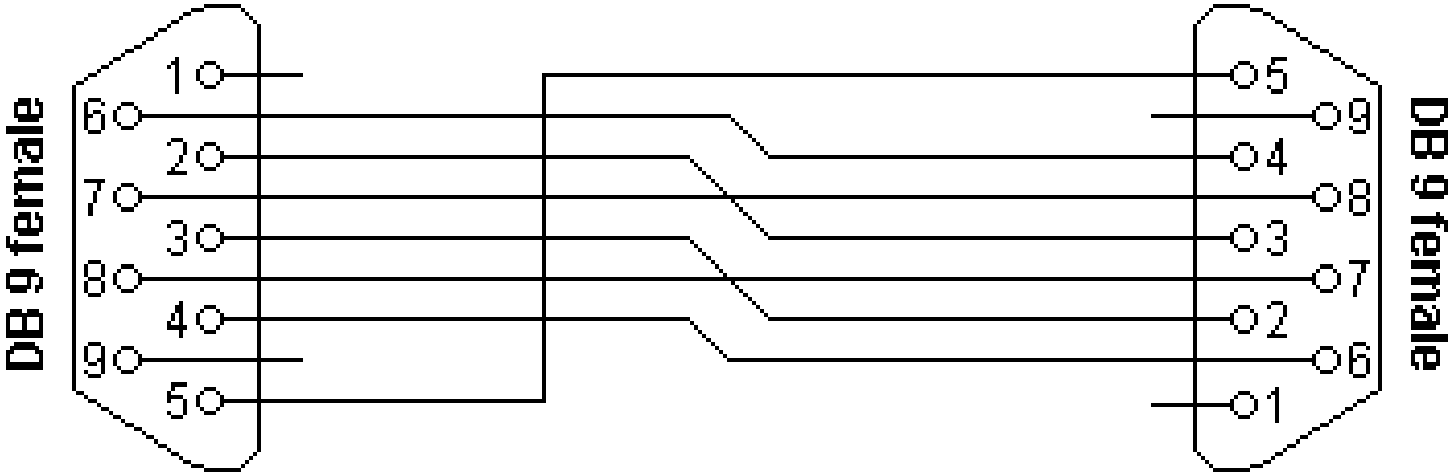
DSR - Data Set Ready - gotowość modemu

DTR - Data Terminal Ready - gotowość terminala

RTS - Request to Send Data - żądanie wysyłania

CTS - Clear to Send - gotowość wysyłania

Pełny kabel null-modem

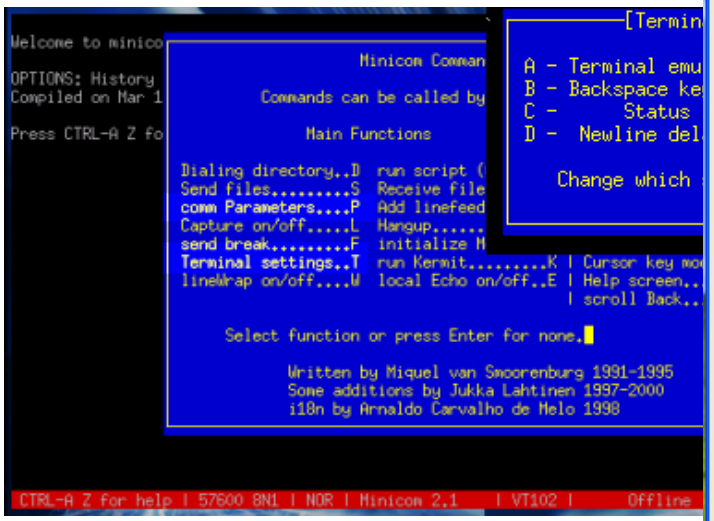


Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
4	6	DTR → DSR
5	5	Signal ground
6	4	DSR ← DTR
7	8	RTS → CTS
8	7	CTS ← RTS

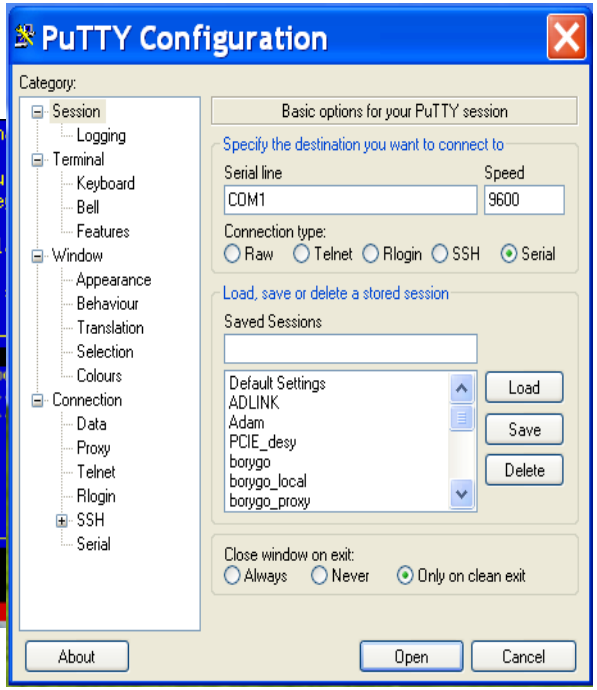
Programy do komunikacji z wykorzystaniem standardu EIA RS232

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)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

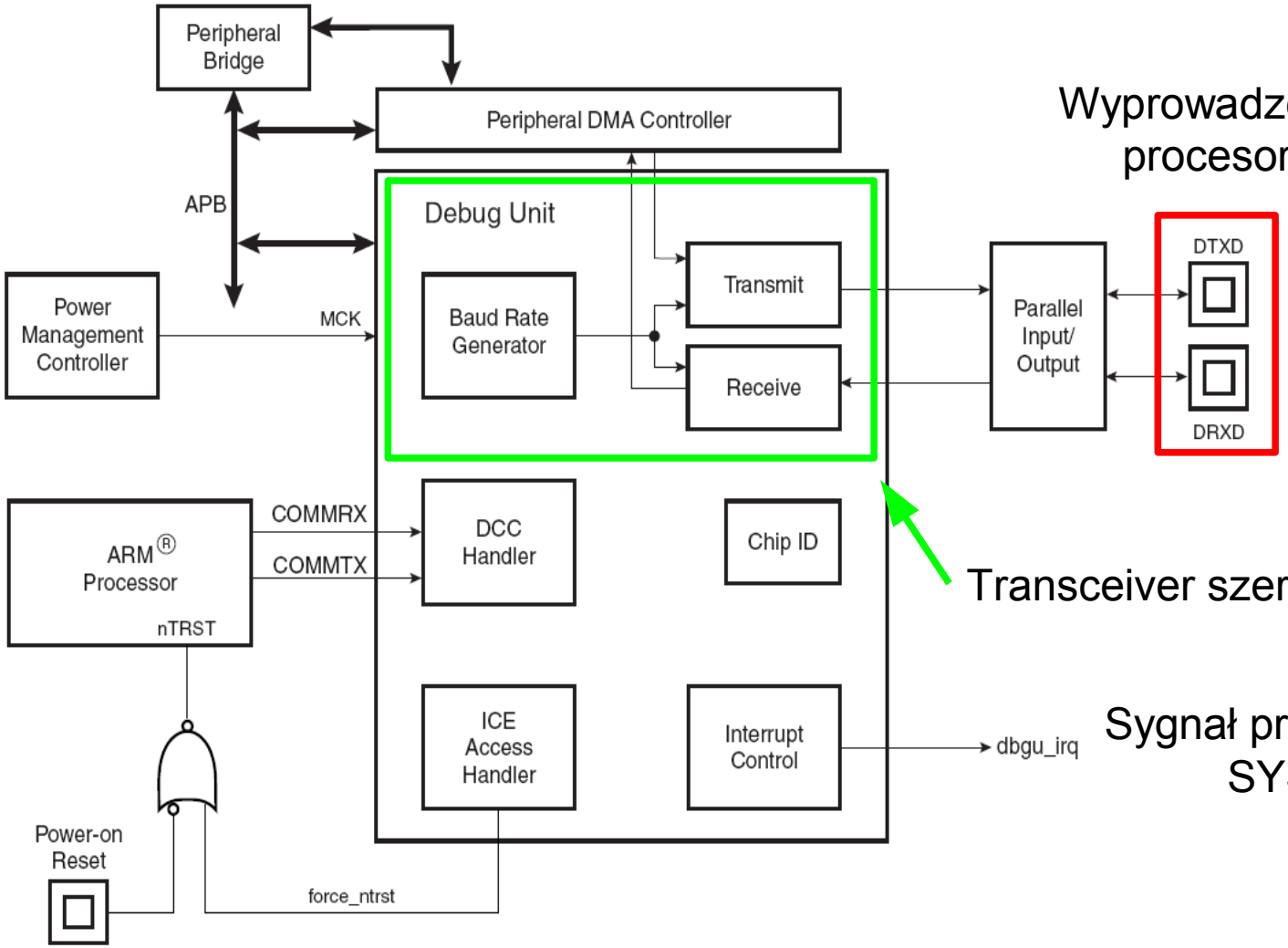
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

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

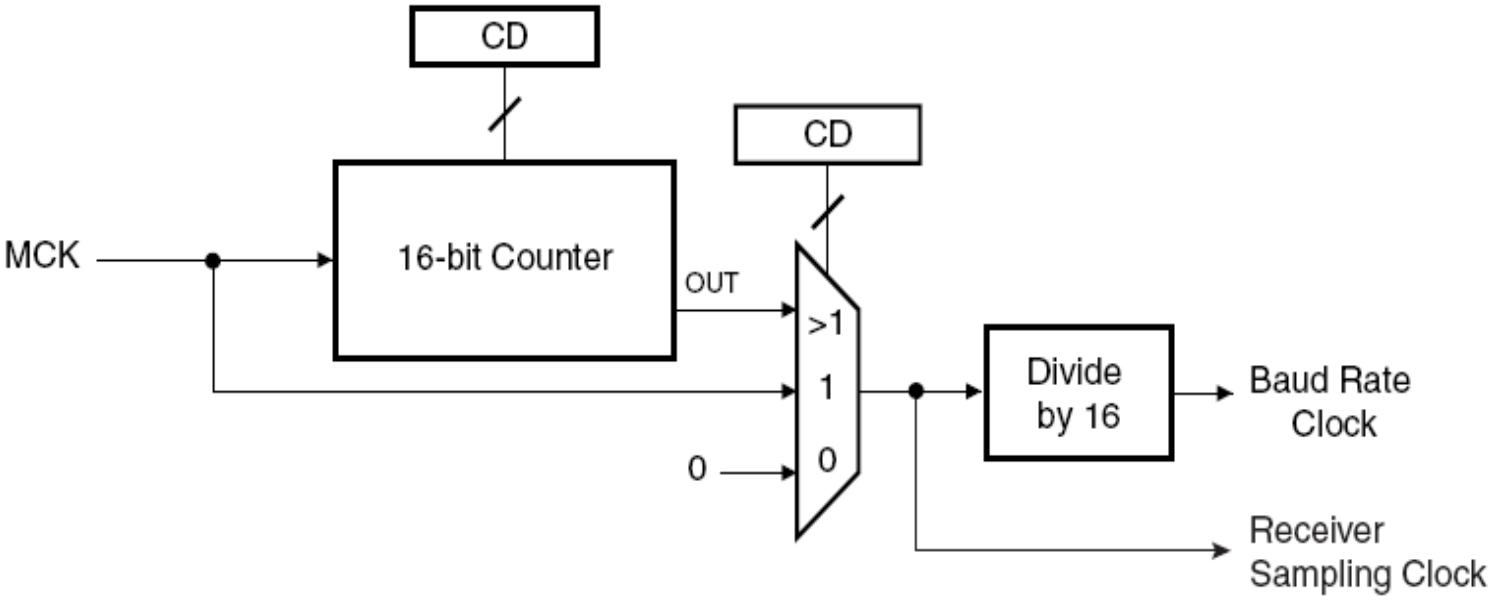


Wyprowadzenia procesora

Transceiver szeregowy

Sygnał przerwnia SYS

Szybkość transmisji



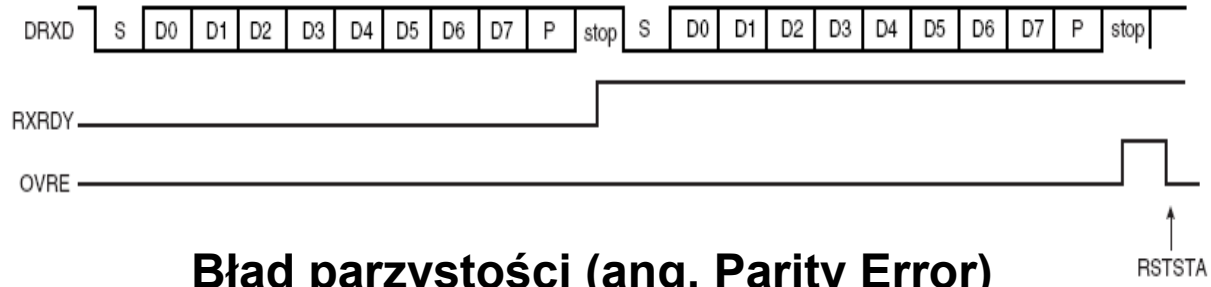
Generator sygnału zegarowego odpowiedzialnego za szybkość transmisji (ang. Baud Rate).

Szybkość transmisji danych wyrażona jest wzorem:

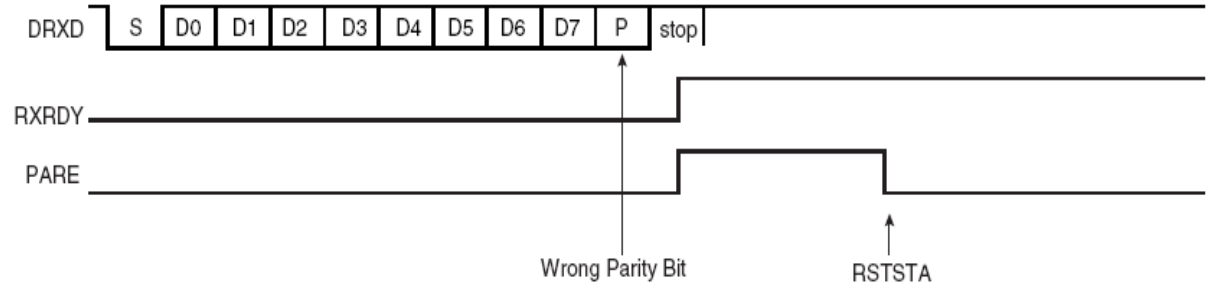
Baud Rate = MCK / (16 x CD), gdzie CD (Clock Divisor) jest polem rejestru DBGU_BRGR

Błędy podczas transmisji danych

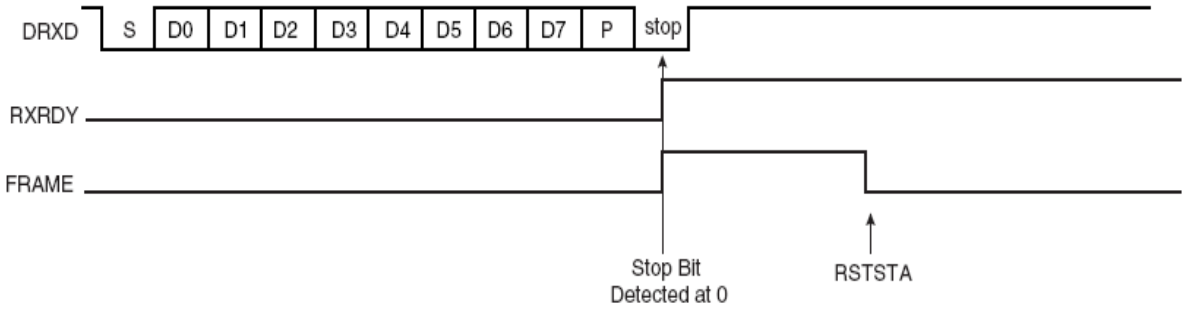
Przepełnienie bufora odbiorczego BGU_RHR (ang. Buffer Overflow)



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... )
}
```

AT91SAM9263 – USART

(rozdział 34)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

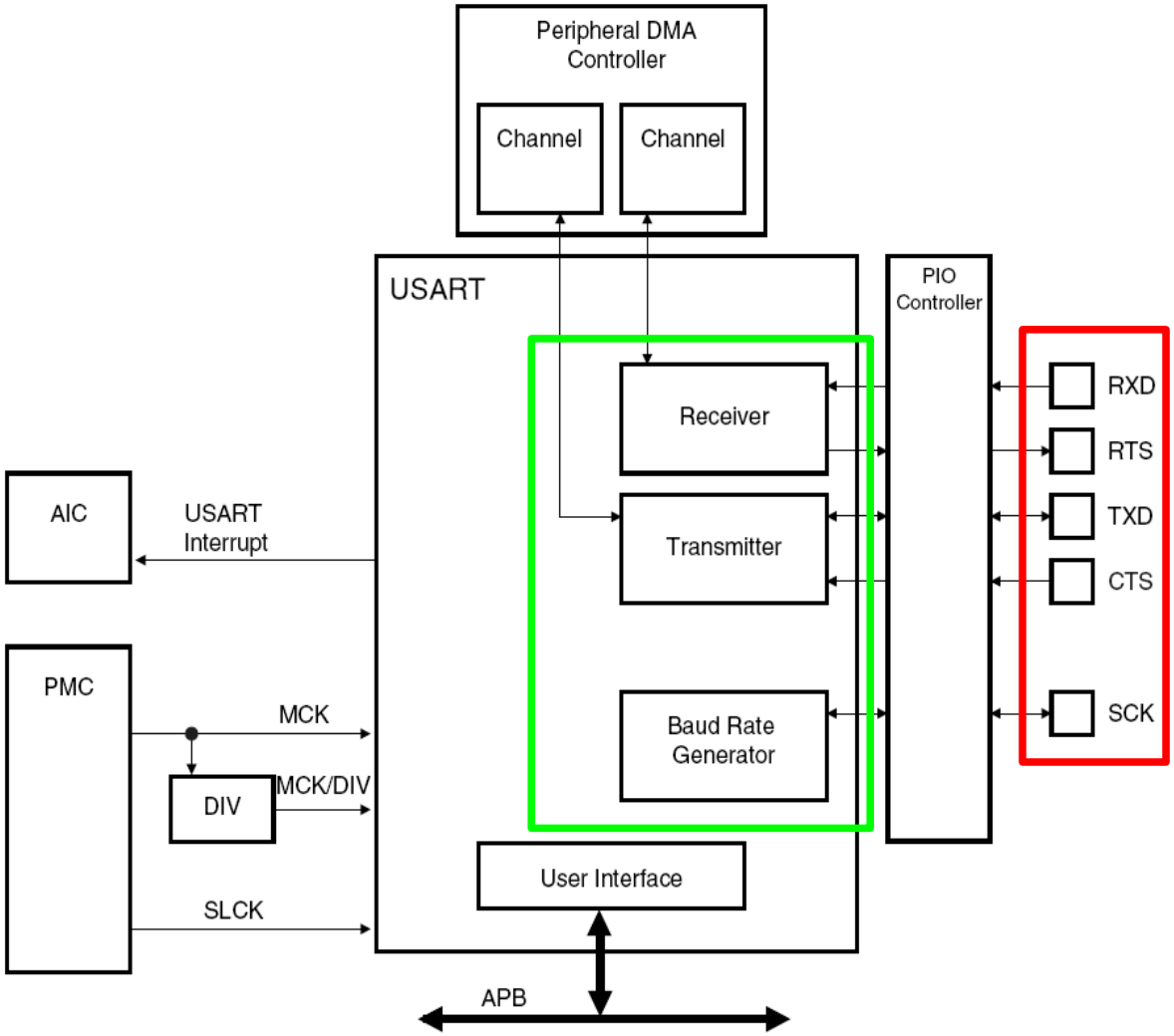
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Port szeregowy USART

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

- Asynchroniczna lub synchroniczna 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



Zakres przedmiotu

- Systemy mikroprocesorowe, systemy wbudowane
- Laboratorium
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Interfejsy w systemach wbudowanych
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

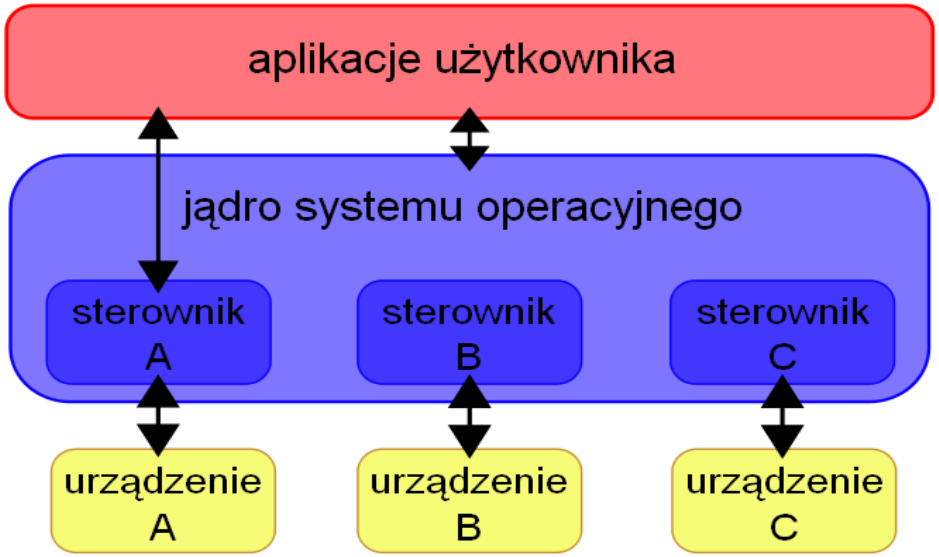
Sterowniki urządzeń peryferyjnych

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

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.**

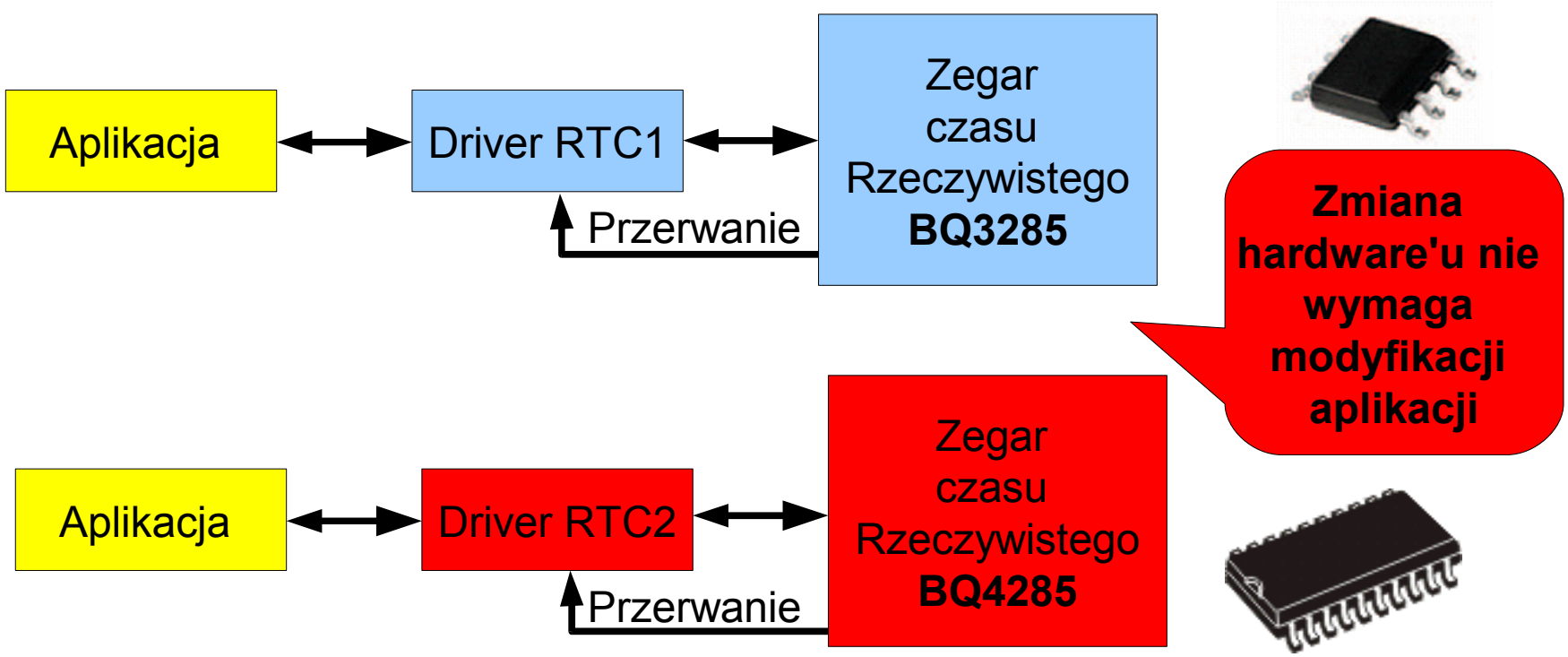
W przypadku systemów wbudowanych **aplikacje mogą się bezpośrednio odwoływać do urządzeń,** czasami trudno jest odróżnić aplikację od sterownika.

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... Funkcja 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.

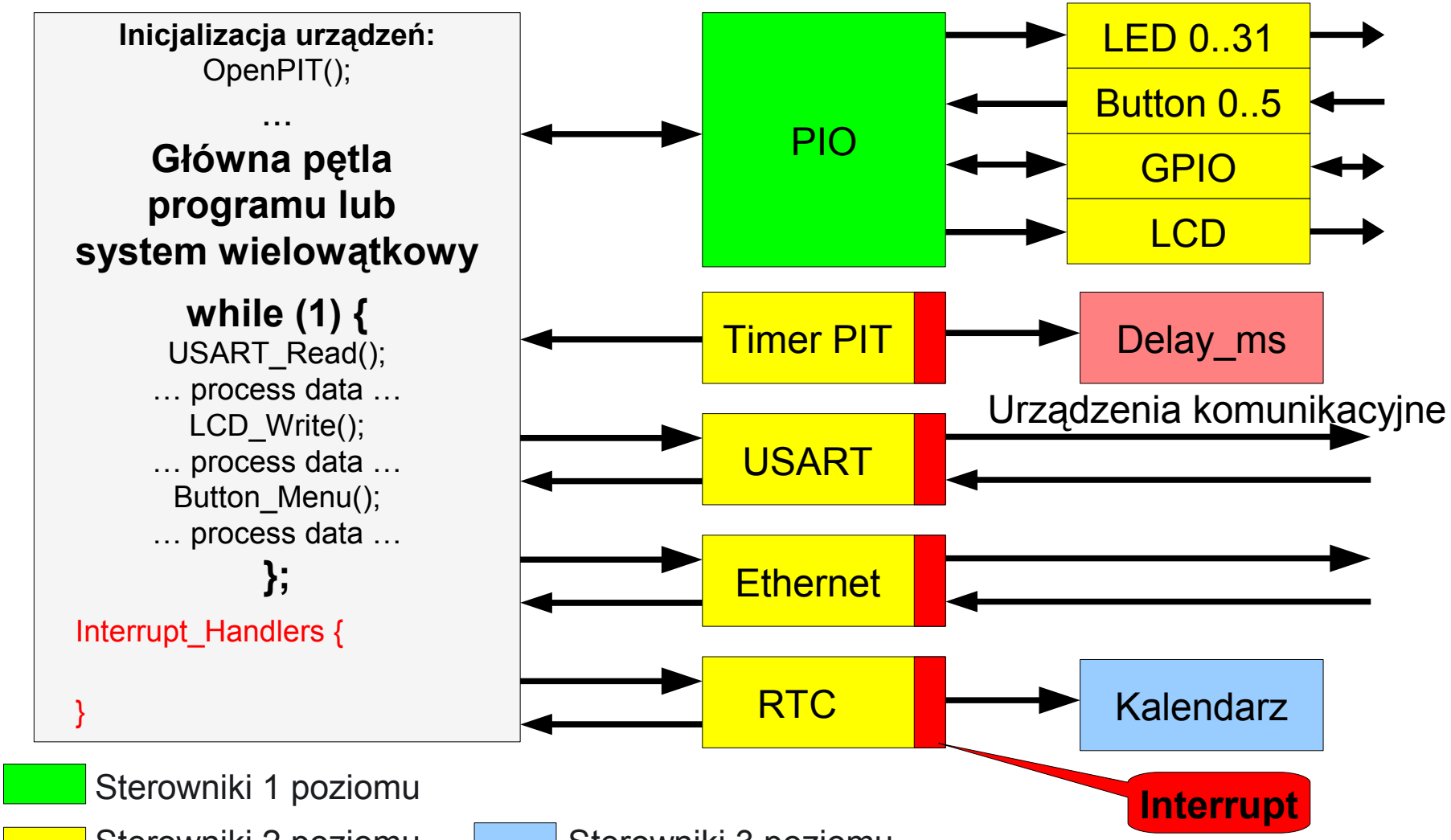
Device_WriteString () - funkcja wykorzystywana do zapisywania ciągu znaków do urządzenia. Funkcja korzysta z funkcji **Device_Write()** zapisującej pojedynczy znak. Funkcja dziedziczy własności blokujące po funkcji niższego poziomu.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

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:

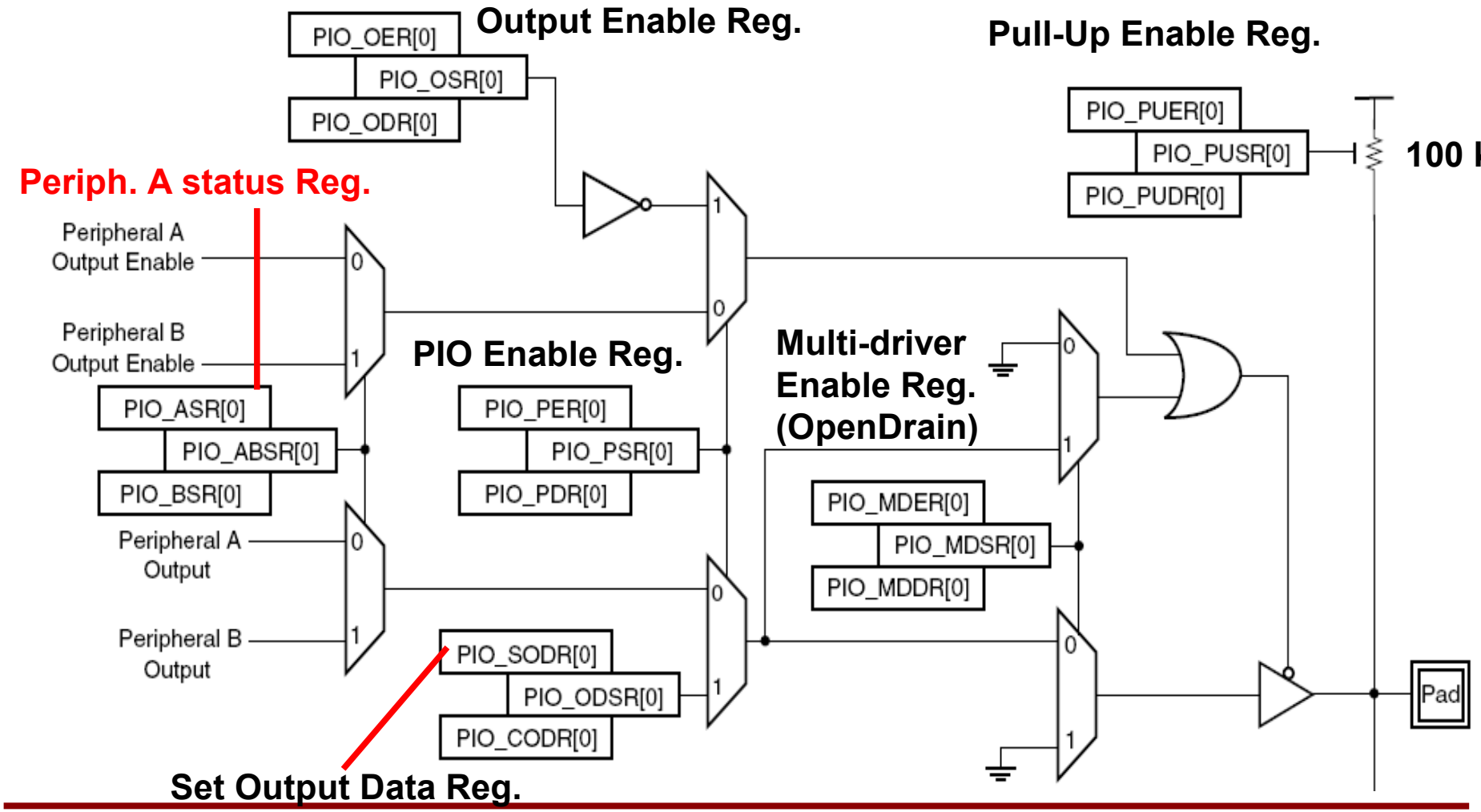
void **PIO_EnablePullUp** (unsigned int *RegistersPointer, unsigned int PortMask);

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

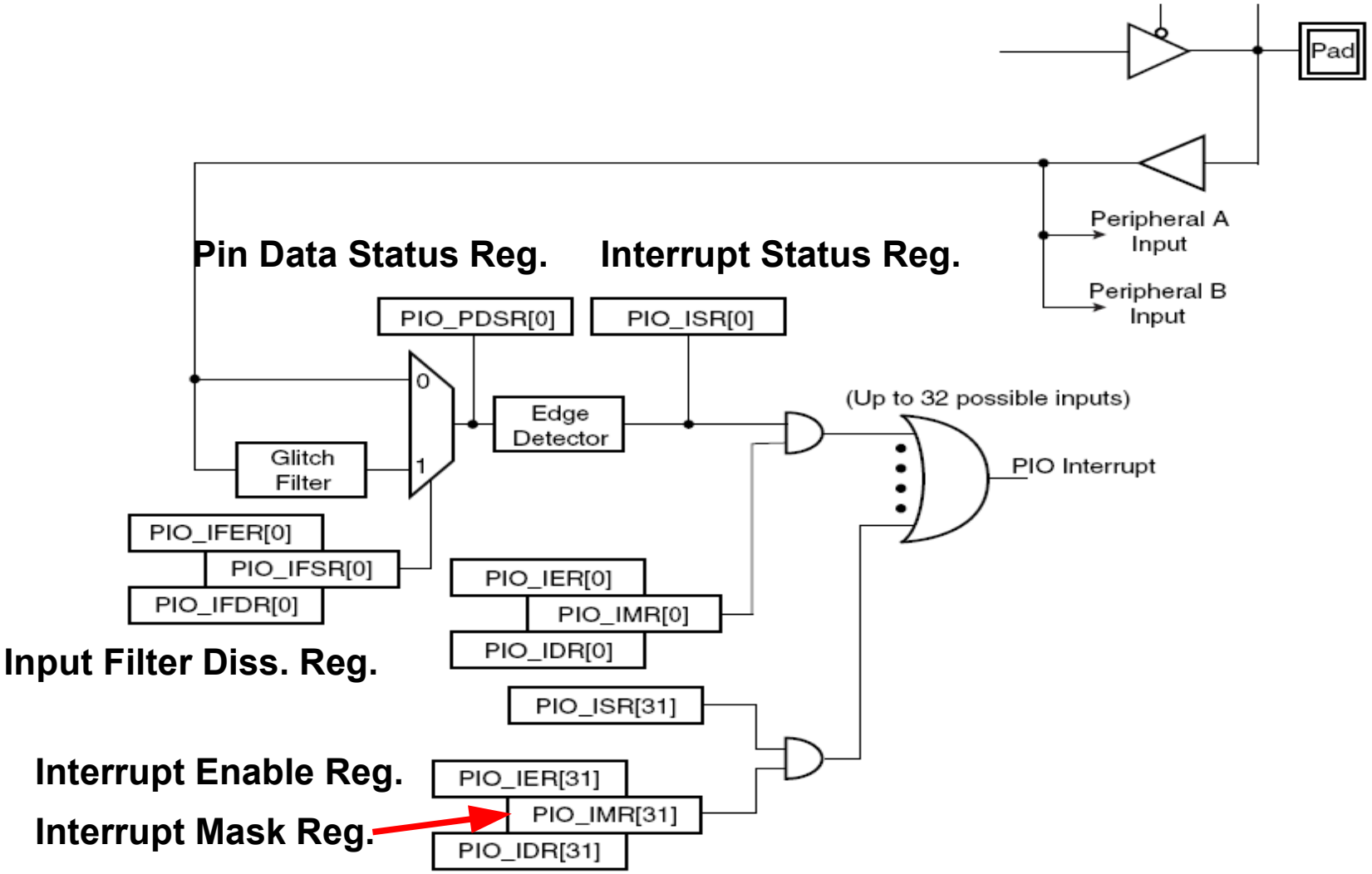
unsigned int **PIO_StatusPullUp** (unsigned int *RegistersPointer, unsigned int PortMask);

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)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

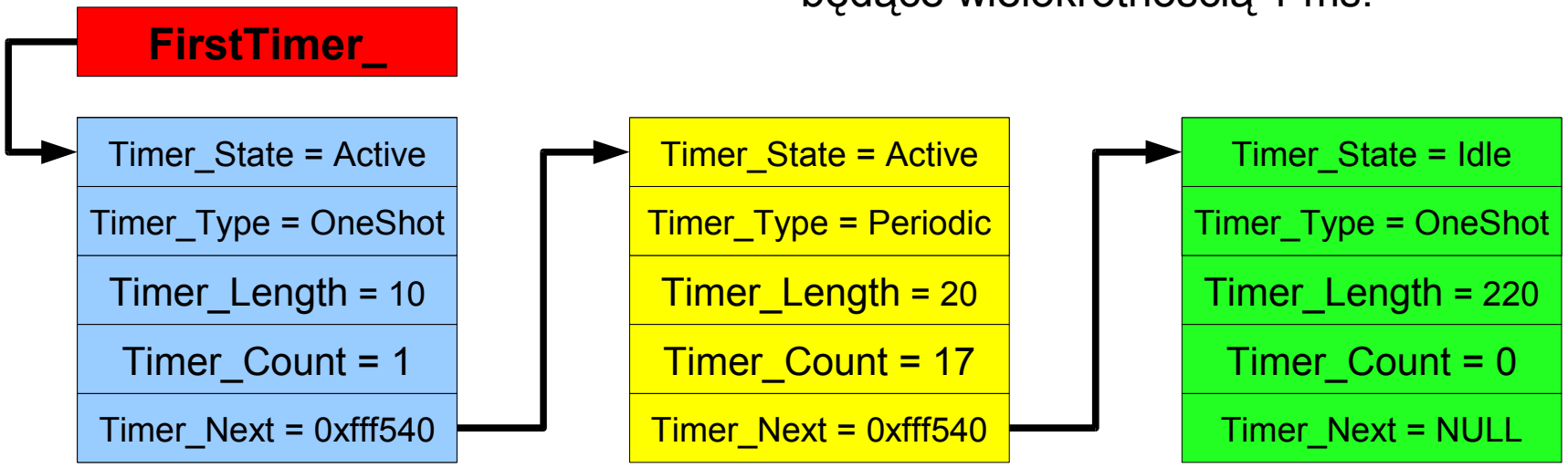
Sterownik obsługujący kilka timer'ów

```
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)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Struktura pliki startowego

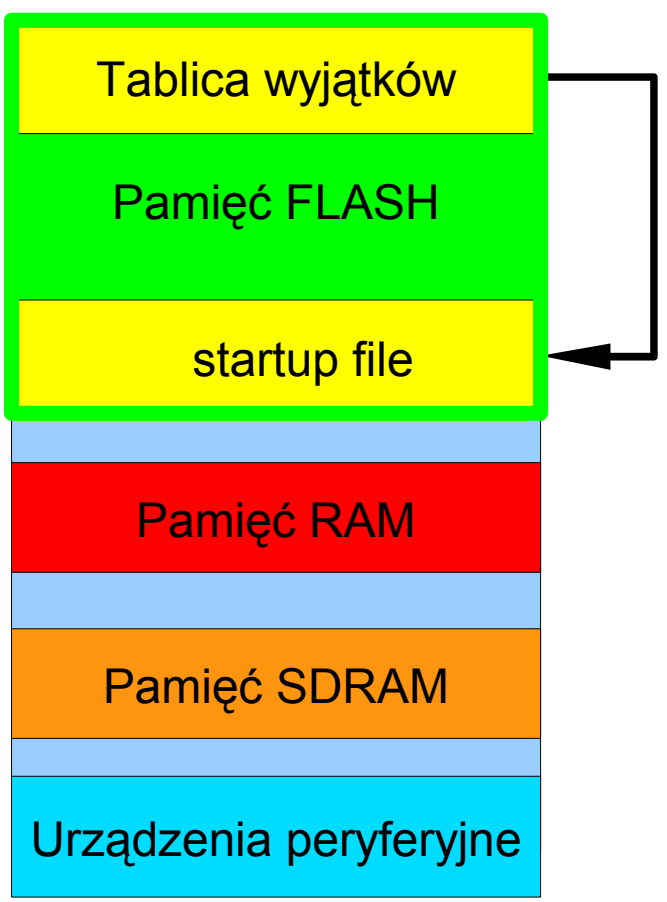
Program startowy uruchamiany jest zaraz po **sygnale resetu** w celu konfiguracji podstawowych zasobów procesora. Plik startowy jest zwykle napisany w języku asemblera ze względu na odwołania do specyficznych zasobów procesora (dostępnych z poziomu asemblera) **i uruchamiany przed** programem napisanym 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.

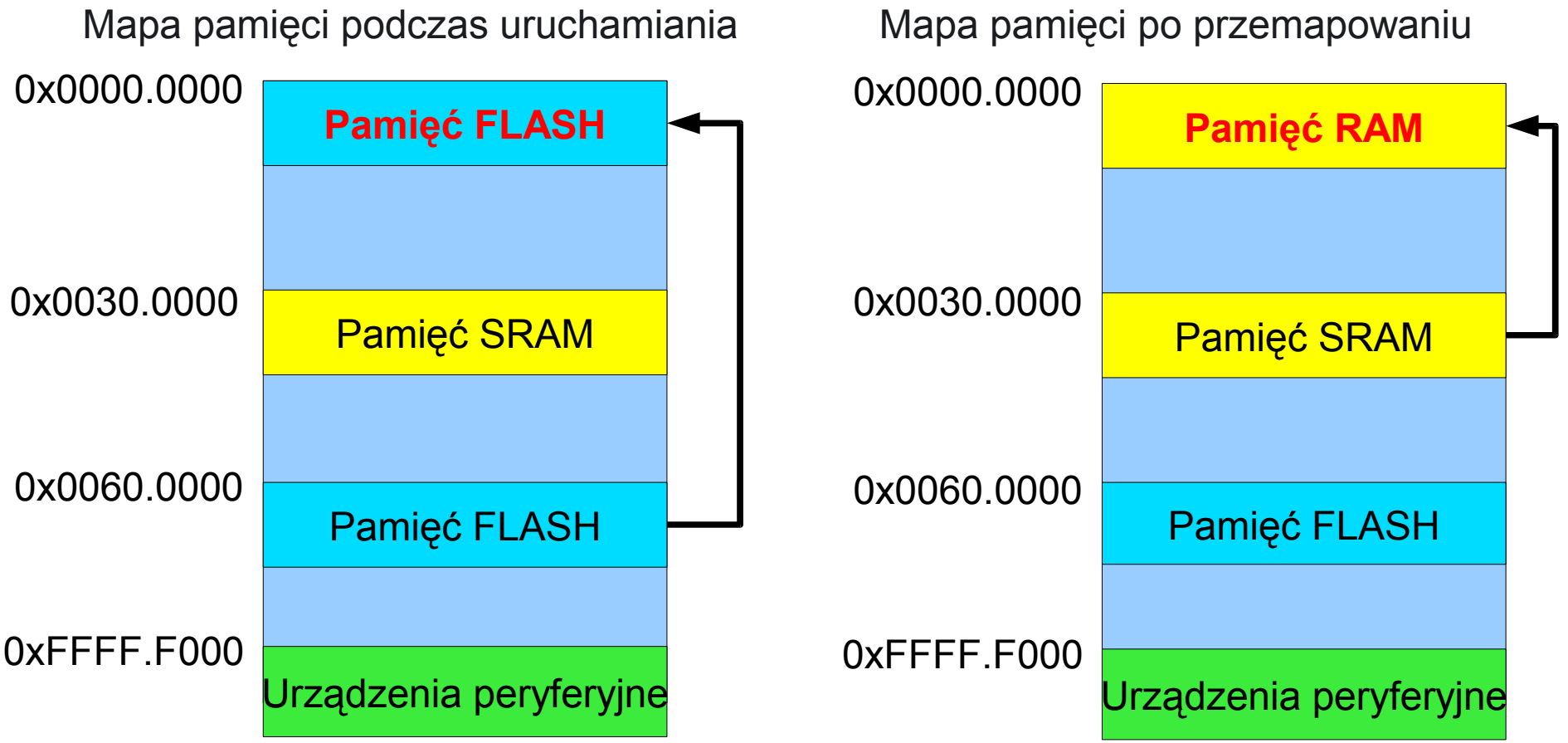
```
...
...
_branch_main:
        ldr      r0, =main
        mov      lr, pc
        bx       r0
...
...
void main (void) {
                While (1)
                {

                        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 (User, IRQ, FIQ,...),
- Odblokowanie wejścia NRST dla zewnętrznego sygnału reset.

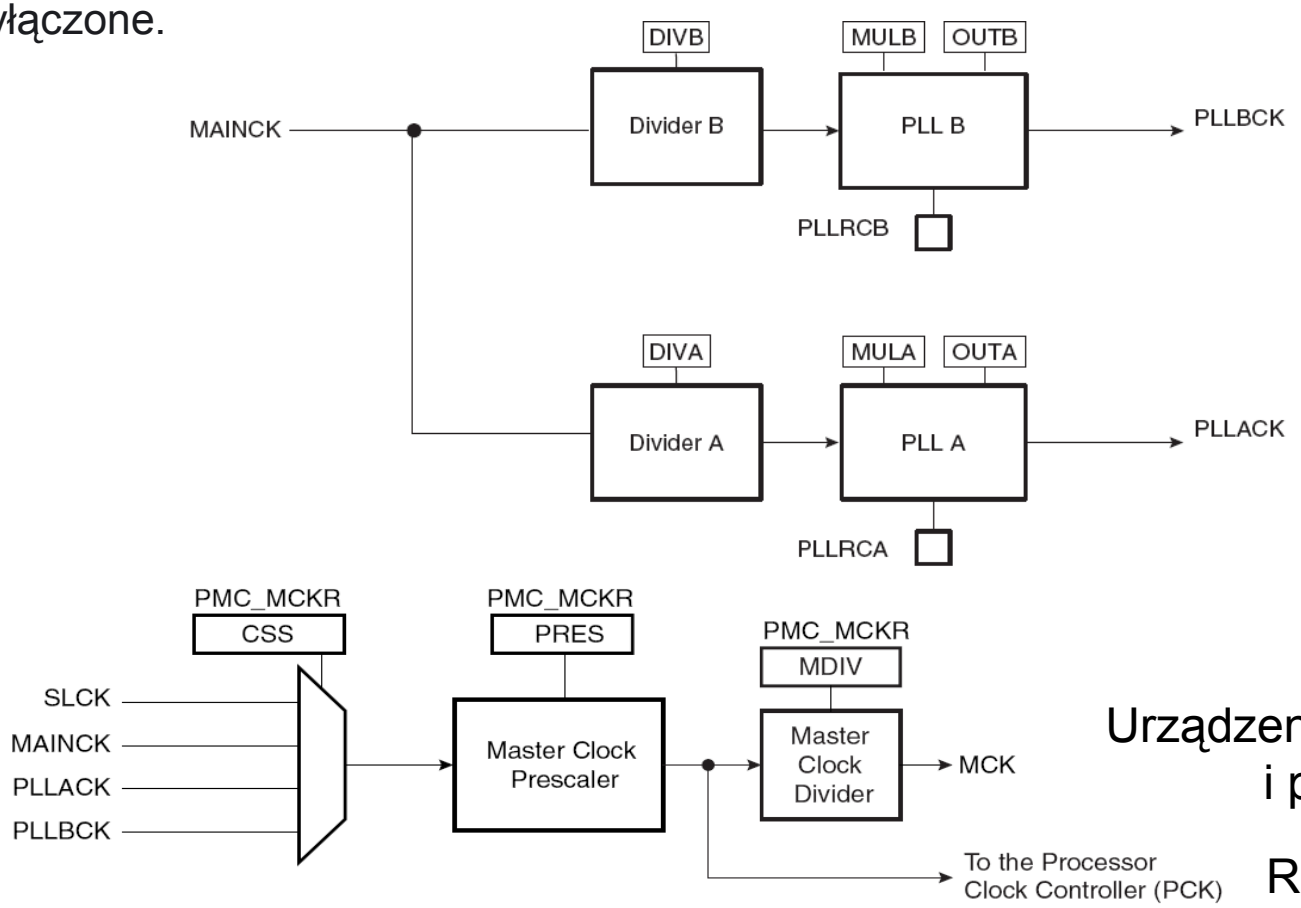
Podmiana pamięci po uruchomieniu



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 = 32 768 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

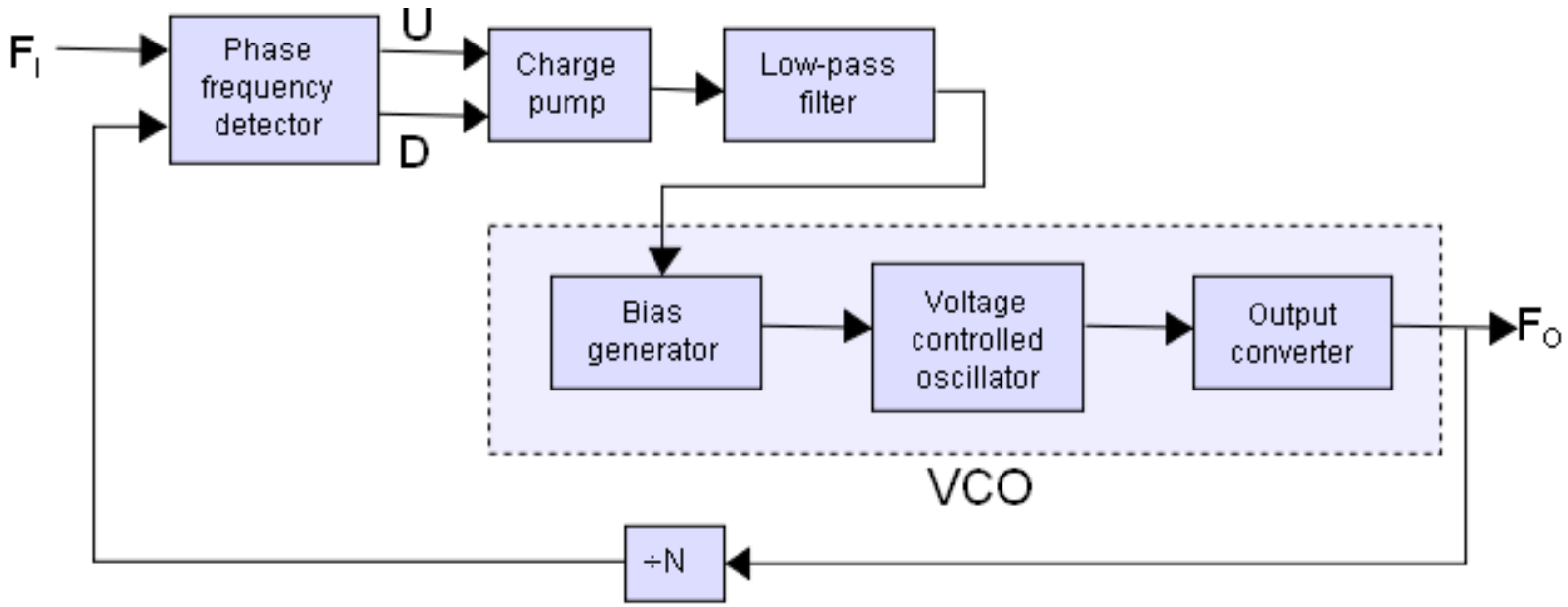
Generator z pętlą PLL (1)

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 (Voltage Controlled Oscillator),
- pętli sprzężenia zwrotnego, w której występuje **dzielnik** częstotliwości (**mnożnik** dla częstotliwości sygnału wyjściowego).

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. Należy odczekać, aż częstotliwość 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 plików lowlevel.c oraz startup.S

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Asembler procesorów ARM

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Asembler procesora ARM

Język asemblera (ang. assembler) to język programowania niskiego poziomu, którego pojedyncze polecenie odpowiada pojedynczemu rozkazowi procesora. Język powstał na bazie języka maszynowego danego procesora poprzez zastąpienie kodów operacji ich mnemonikami.

Cechy asemblera w trybie ARM:

- Instrukcje procesora RISC,
- Operacje na pamięci przy wykorzystaniu instrukcji load-store,
- Instukcje o stałej długości (32-bit),
- Wszystkie instrukcje mogą być wykonywane warunkowo, wykonanie instrukcji bezwarunkowych z warunkiem always – AL.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Instrukcje asemblera procesora ARM

Grupa	Mnemonik	Rozwinięcie mnemonika	Opis
Arytmetyczne	ADD ADC	*add / add with carry*	dodawanie / dodaw. z uwzględnieniem bitu carry
	SUB SBC	*substract / substr. with carry*	odejmowanie / odejm. z uwzględnieniem bitu carry
	RSB RSC	*revers substract / rewers substract with carry*	odejmowanie w odwrotnej kolejności / / odejm. w odwr. kol. z uwzględnieniem bitu carry
	CMP CMN	*compare / compare negative*	porównanie / porówn. ze zmienionym znakiem arg2
Logiczne	AND BIC	*and / bit clear (and not)*	iloczyn logiczny / zerowanie bitów
	ORR EOR	*or / exor*	suma logiczna / różnica symetryczna
	TST TEQ	*test / test equivalence*	test / test identyczności
Mnożenia	MUL MLA	*multiply / multiply-accumulate*	mnożenie / mnożenie z dodawaniem
	UMULL SMULL	*unsigned / signed multiply*	mnożenie bez znaku / mnożenie ze znakiem
	UMLAL SMLAL	*unsigned / signed multiply-accumulate*	mnożenie z dodawaniem bez znaku / moż. z dodaw. ze znakiem
Skoki	B	*branch*	rozgałęzienie (skok)
	BL	*branch with link*	rozgałęzienie (skok) z zachowaniem rej. PC
	BX	*branch and exchange*	rozgałęzienie (skok) ze zmianą trybu ARM/Thumb
Przesłań	MOV MVN	*move / move not*	przesłania pomiędzy rejestrami oraz ładowanie stałych do rejestrów / j.w. z negacją
	LDR STR	*load / store register*	przesłania rejestru z/do pamięci
	LDM STM	*load / store multiply register*	przesłania wielu rejestrów z/do pamięci
	SWP	*swap register and memory*	wymiana zawartości rejestru i pamięci
	MRS MSR	*move xPSR to / from register*	przesłania pomiędzy rejestrami statusowymi a rejestrami
Pozostałe	SWI	*software interrupt*	przerwanie programowe
Rozkazy opcjonalnych koprocesorów	MRC MCR	*move coprocessor – register*	przesłania rdzeń – koprocesor
	LDC STC	*move coprocessor – memory*	przesłania pamięć – koprocesor
	CDP	*coprocessor data operation*	instrukcja koprocesora

Tabela warunków dla instrukcji asemblera

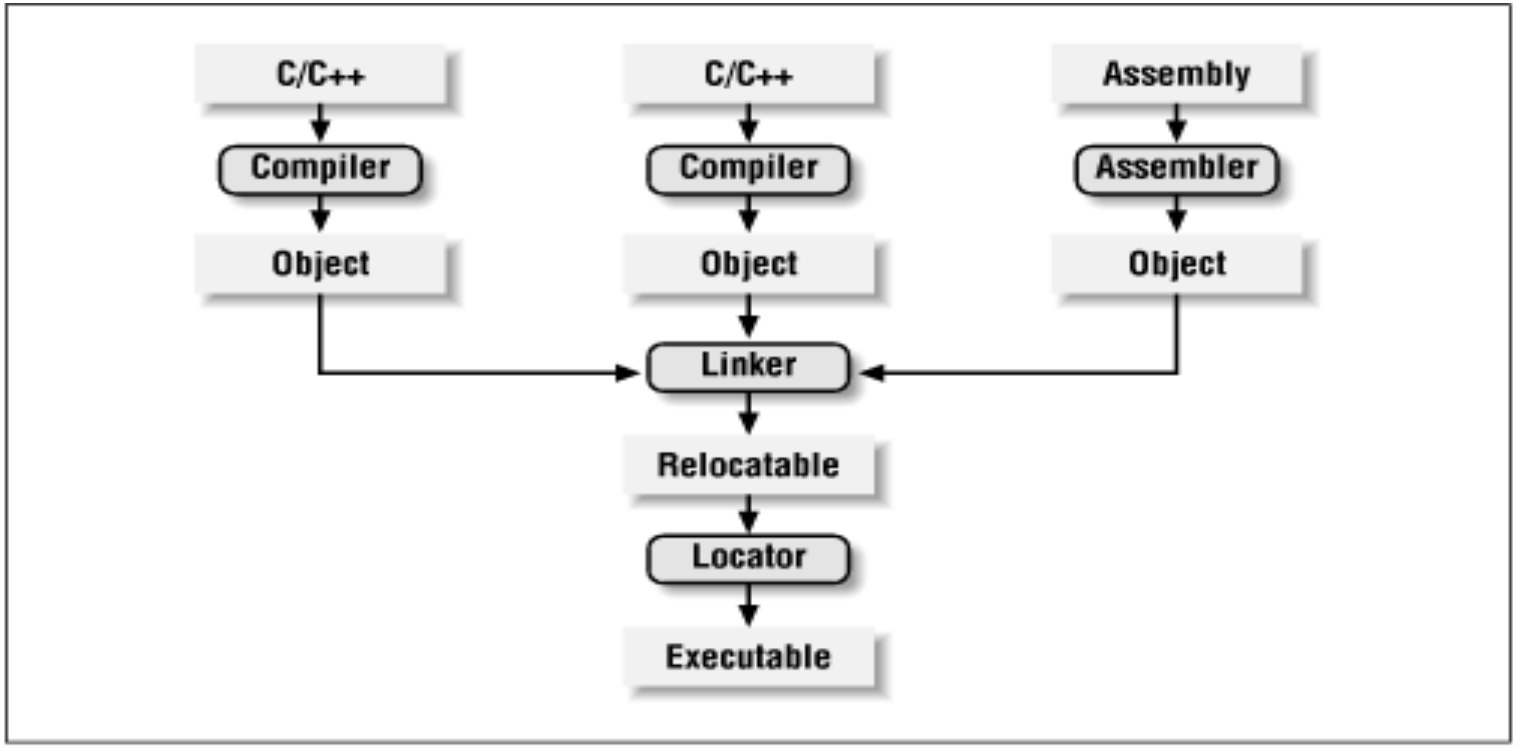
Mnemo-nik	Rozwinięcie mnemonika (ang.)	Warunek	Stan flag
EQ	*equal*	równy	Z=1
NE	*not equal*	nie równy	Z=0
CS	*carry set (unsigned higher or same)*	ustawiona flaga przeniesienia carry; większy lub równy (liczby bez znaku)	C=1
CC	*carry clear (unsigned lower)*	wyzerowana flaga przeniesienia carry; mniejszy (liczby bez znaku)	C=0
MI	*negative (minus)*	ujemny	N=1
PL	*positive or zero (plus)*	dodatni lub zerowy	N=0
VS	*overflow set*	ustawiona flaga przepełnienia	V=1
VC	*overflow clear*	wyzerowana flaga przepełnienia	V=0
HI	*unsigned higher*	większy (liczby bez znaku)	C=1 and Z=0
LS	*unsigned lower or same*	mniejszy lub równy (liczby bez znaku)	C=0 or Z=1
GE	*greater or equal*	większy lub równy	N=V
LT	*less then*	mniejszy	N<>V
GT	*greater then*	większy	Z=0 and (N=V)
LE	*less than or equal*	mniejszy lub równy	Z=1 or (N<>V)
AL	*always*	zawsze (mnemonik można pominąć)	bez znaczenia

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Proces kompilacji programu

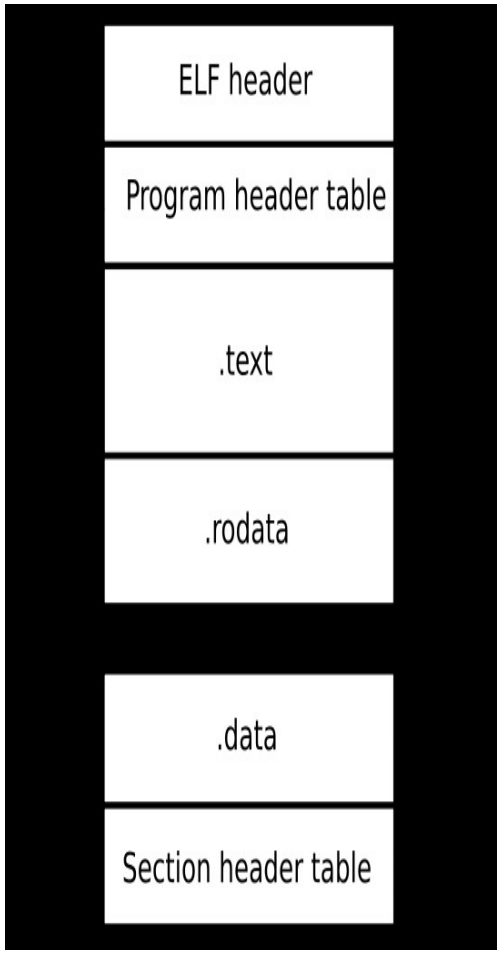


- 1 faza – kompilacja plików źródłowych → plik binarny, relokowalny
- 2 faza – linkowanie plików relokowalnych → plik binarny, relokowalny (.o)
- 3 faza – przypisanie adresów → plik binarny, wykonywalny (.elf, .coff, .a)

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)
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

„Procesory ARM w systemach wbudowanych”
”Programowanie procesorów ARM”

Prezentacja jest współfinansowana przez
Unię Europejską w ramach
Europejskiego Funduszu Społecznego w projekcie pt.

„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”

Prezentacja dystrybuowana jest bezpłatnie