STM32 dla początkujących (i nie tylko)

Komunikacja za pomocą UART

W piątej części kursu poświęconego mikrokontrolerowi STM32F i Panelowi Edukacyjnemu zajmę się interfejsem UART. Pod tą nazwą kryje się powszechnie wykorzystywany i mający długą historię mechanizm transmisji danych pomiędzy mikrokontrolerem a urządzeniami zewnętrznymi.

UART w STM32 – podstawowe wiadomości

Jeżeli kontroler pracujący w urządzeniu musi komunikować się z innymi urządzeniami np. komputerem PC, najwygodniej zrobić to wykorzystując jeden z przyjętych standardów komunikacyjnych. Interfejs szeregowy UART umożliwia dwukierunkową wymianę danych przesyłając dane kolejno, bit po bicie. Skrót nazwy UART oznacza *Universal Asynchronous Receiver-Transmitter*, co po polsku oznacza uniwersalny asynchroniczny odbiornik-nadajnik. Jednak współczesne UARTy mają też inne tryby transmisyjne, więc ich nazwa uległa modyfikacji – dodano tryb pracy synchronicznej, więc nazwa UART zmieniła się w USART (*Universal Synchronous-Asynchronous Receiver-Transmitter*, co oznacza uniwersalny synchroniczny-asynchroniczny odbiornik-nadajnik).

Interfejs UART często jest utożsamiany z RS232, jednak trzeba pamiętać, że pomimo identycznej ramki danych, pracuje z wykorzystaniem napięcia o innych poziomach logicznych oraz, że o tym, że typowo w UART na próżno szukać sygnałów tzw. handshakingu, to jest RTS, CTS, DTR itd. Typowo USARTy mikrokontrolerów używają do transmisji danych jedynie trzech linii:

1. TxD – linia danych wysyłanych. W urządzeniu współpracującym powinna być połączona z linią odbioru danych.
2. RxD – linia odbieranych danych. W urządzeniu współpracującym powinna być połączona z linią transmisji danych.
3. GND – linia masy, ustalająca poziom odniesienia dla połączonych urządzeń.

Co ciekawe, interfejs USART w mikrokontrolerach STM32 ma również wyprowadzone dwie linie służące do sprzętowej kontroli przepływu:

- CTS (*Clear To Send*) – wejście sygnału sygnalizującego gotowość urządzenia współpracującego do wysyłania danych. W urządzeniu współpracującym ta linia jest połączona z RTS (*Ready To Send*).
- RTS – wyjście sygnału sygnalizującego gotowość urządzenia do wysyłania danych do urządzenia współpracującego. Po drugiej stronie sygnał powinien być dołączony do linii CTS.

W mikrokontrolera STM32 sygnały interfejsu USART mają poziomy L=0 oraz H=3,3V. Po zastosowaniu prostych konwerterów można przekształcić wyjście USART w port RS232, a nawet przesyłać dane do komputera PC.

Zależnie od typu mikrokontrolera można wykorzystać dwa lub więcej portów USART. Procedury inicjacji każdego portu są identyczne. Kompletne zestawienie możliwości interfejsu USART mikrokontrolera znajduje się w dokumentacji technicznej *CD00171190.pdf* w paragrafie 27.2 „USART main features".

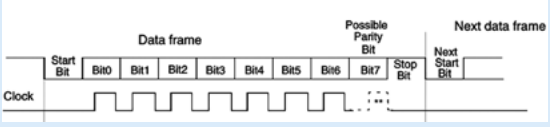
USART – ramka transmsji

Jak wynika z nazwy, transmisja za pomocą USART może być synchroniczna lub asynchroniczna. W pierwszym wypadku transmisja kolejnych bitów danych jest synchronizowana impulsami zegarowymi i jest potrzebna dodatkowa linia. W drugim wypadku linia zegarowa nie jest używana. Mikrokontrolery STM32 mogą obsługiwać oba rodzaje transmisji, choć nieporównanie częściej wykorzystuje się transmisję asynchroniczną.

Aby strona nadawcza i odbiorcza jednakowo interpretowały przesyłany sygnał, musi się to dziać zgodnie z ustalonymi regułami. Te reguły to budowa ramki zawierającej pojedynczy bajt i ustalona szybkość transmisji, co jest szczególnie ważne dla trybu asynchronicznego. Budowę pojedynczej ramki danych przesyłanej za pomocą USART pokazano na **rysunku 1**.

Każdą transmisję kolejnej ramki danych rozpoczyna bit startu, który zawsze ma poziom niski. Następnie przesyłane są kolejno bity danych począwszy od bitu najmłodszego (numer 0) do najstarszego. W mikrokontrolerze STM32 można wybrać transmisję z 8 lub 9 bitami danych. Poziom wysoki oznacza transmisję logicznej „1" a poziom niski transmisję logicznego „0". Jeżeli będzie wykorzystywana kontrola parzystości, to informacja o parzystości lub jej braku będzie kodowana na pozycji najstarszego bitu danych. Ramkę kończy bit stopu, który ma zawsze poziom niski.

Przy transmisji synchronicznej na dodatkowej linii transmitowane są impulsy zegarowe synchronizujące odbiór danych. Przy transmisji asynchronicznej linia zegarowa jest niepotrzebna.



Rysunek 1. Budowa ramki transmitowanej za pomocą UART

Tabela 1. Standardowe szybkości transmisji, które można zaprogramować w mikrokontrolerze						
numer	f PCLK = 36 MHz			f PCLK = 72 MHz		
	Szybkość Kbps	Rzeczywista szybkość	Błąd %	Szybkość Kbps	Rzeczywista szybkość	Błąd %
1	2400	2400	0	2400	2400	0
2	9600	9600	0	9600	9600	0
3	19200	19200	0	19200	19200	0
4	57,6	57,6	0	57,6	57,6	0
5	115,2	115,384	0,15	115,2	115,2	0
6	230,4	230,769	0,16	230,4	230,769	0,16
7	460,8	461,538	0,16	460,8	461,538	0,16
8	921,6	923.076	0,16	921,6	923.076	0,16
9	2250	2250	0	2250	2250	0
10	4500	-	-	4500	4500	0

Szybkość transmisji

Szybkość transmisji określa czas trwania pojedynczego bitu, a w konsekwencji czas transmisji pojedycznej ramki danych. Interfejs w mikrokontrolerach STM32 umożliwia ustawienie dowolnej prędkości transmisji. Przyjęło się jednak używać standardowych wartości, co np. ułatwia procedury automatycznej detekcji transmisji. W **tabeli 1** umieszczono niektóre standardowe szybkości możliwe do zaprogramowania w mikrokontrolerze:

W mikrokontrolerach STM32F wewnętrzne układy portu USART1 można taktować przebiegiem zegara systemowego o częstotliwości 72 MHz, a pozostałe częstotliwością o połowę mniejszą. Tabela zestawia nominalne standardowe szybkości transmisji z możliwymi do ustawienia najbliższymi im wartościami oraz wartość maksymalnego błędu wynikającego z różnicy pomiędzy prędkością uzyskaną, a wymaganą przez standard.

Pozostałe ustawiane parametry transmisji

Oprócz samej szybkości mikrokontrolery STM32F pozwalają na ustawienie kilku dodatkowych parametrów pracy USART:

- Liczby bitów danych (8 lub 9).
- Włączenie kontroli parzystości. Jest to prosty sposób kontroli poprawności przesłanych danych. W trybie „evenparity" bit parzystości jest, gdy liczba jedynek danych ramki jest nieparzysta. W trybie „oddparity" bit jest ustawiany, gdy liczba jedynek danych jest parzysta. Bit przesyłany jest na najstarszej pozycji.
- Czas trwania bitu stopu: 0,5, 1, 1,5 lub 2 okresy transmisji pojedynczego bitu danych.

Źródła przerwań

Interfejs USART może być źródłem kilku przerwań. Ich wykorzystanie pozwala na uwolnienie programu głównego od stałej, czasochłonnej kontroli stanu USART. Pozwala także na pracę w trybie wieloportowym. Najczęściej używane źródła przerwań od USART wymieniono w **tabeli 2**.

Biblioteczne funkcje obsługi USART-a

Biblioteka *STM32F10x Standard Peripherals Firmware Library* zawiera cały zestaw funkcji związanych z ustawieniem parametrów i obsługą interfejsów USART. W tym celu odnaleźć i kliknąć plik pomocy *stm32f10x_stdperiph_lib_um.chm*. W polu wyszukiwania *Index* należy wpisać USART. Wykaz funkcji znajduje się w sekcji *USART Exported Functions*.

Obsługę interfejsu USART można zrealizować albo poprzez bezpośredni nadzór programu głównego nad transmisją i odbiorem pojedynczych bajtów w trybie tzw. polingu (odpytywania), albo wykorzystując system przerwań. Ponieważ drugi sposób jest zdecydowanie bardziej efektywny w opisie skupię się na obsłudze interfejsu szeregowego za pomocą przerwań z wykorzystaniem funkcji bibliotecznych/

Typowa inicjacja i obsługa dwustronnej transmisji asynchronicznej portem USART składa się z następujących kroków:

- Inicjacji linii portów pełniących funkcje TxD i RxD dla wybranego portu USART.
- Dołączenia wewnętrznego zegara do obwodów linii i interfejsu USART.
- Ustawienia parametrów interfejsu USART.
- Ustawienia bitu włączającego przerwanie odbioru (przerwanie nadawania powinno pozostać wyłączone).
- Utworzenia procedur obsługi przerwań.
- Włączenia kontrolera przerwań NVIC.

Gdy zachodzi potrzeba wysłania danych za pomocą USART, należy włączyć przerwanie nadawania. Procedura obsługi przerwań powinna automatycznie wyłączać przerwanie, gdy zakończone zostanie wysyłanie wszystkich bajtów.

Tabela 2. Najczęściej używane źródła przerwań od USART		
Opis przerwania	Flaga ustawiana po wystąpieniu przerwania	Bit włączający przerwanie
Przerwanie wywoływane zakończeniem wysyłania zawartości rejestru transmisji nadawania	TXE	TXEIE
Przerwanie wywoływane po umieszczeniu w rejestrze odbiorczym odebranego bajtu danych	RXNE	RXNEIE
Przerwanie na skutek nadpisania w rejestrze odbiorczym danej nieodczytanej przez program nową daną odebraną przez USART	ORE	
Przerwanie po wystąpieniu błędu parzystości danej odebranej	PE	PEIE

Przykłady procedur portu USART

Procedura inicjowania portów mających pełnić funkcje wyprowadzeń RxD i TxD jest typowa. Oczywiście, na samym początku korzystając z dokumentacji lub opisywanego w poprzednim odcinku kursu programu narzędziowego *STM32CubeMX* należy określić, które linie mikrokontrolera pełnią funkcję wyprowadzeń RxD i TxD interfejsu USART, który chcemy uruchomić. Przykład będzie dotyczył USART1, którego standardowymi wyprowadzeniami są: RxD – PA.10, TxD – PA.9. W pliku nagłówkowym można wcześniej zadeklarować nazwy zmiennych określających te linie. Wszystko razem może wyglądać jak na **listingu 1**.

Następnie ustawiane są parametry portu. W tym wypadku port USART będzie mógł jednocześnie wysyłać i odbierać. Dane przesyłane będą w formacie 8-bitowym z 1 bitem stopu. Sprzętowa kontrola przepływu i kontrola parzystości są wyłączone. Na początku do układów USART procedura podłącza sygnały zegarowe. Na końcu włączane jest przerwanie odbioru (**listing 2**).

Listing 1. Inicjowanie USART

```
//definicje USART1
#define US-ART_1              USART1
#define USART_1_GPIO          GPIOA
#define USART_1_CLK           RCC_APB2Periph_USART1
#define USART_1_GPIO_CLK      RCC_APB2Periph_GPIOA
#define USART_1_RxPin         GPIO_Pin_10
#define USART_1_TxPin         GPIO_Pin_9
#define USART_1_IRQn          USART1_IRQn
#define USART_1_IRQHandler    USART1_IRQHandler

//inicjowanie i konfigurowanie linii I/O będących
wyprowadzeniami RxD, TxD
void USART1_GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
/* Configure USART1 Rx as input floating */
    GPIO_InitStructure.GPIO_Pin = USART_1_RxPin;
    GPIO_InitStructure.GPIO_Mode =GPIO_Mode_IPU;
    GPIO_Init(USART_1_GPIO, &GPIO_InitStructure);
/* Configure USART1 Tx as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin = USART_1_TxPin;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(USART_1_GPIO, &GPIO_InitStructure);
}
```

Jako parametr wywołania procedury należy podać szybkość transmisji, z którą ma pracować port USART np.

Listing 2. Inicjowanie interfejsu USART

```
//---------------------------------
//procedury inicjowania interfejsu USART1
void USART1_Inicjacja(uint32_t usart_speed)
{
    USART_InitTypeDef USART_InitStructure;
/* USARTy and USARTz configured as follow:
  - BaudRate = usart_speed baud
  - Word Length = 8 Bits
  - One Stop Bit
  - No parity
  - Hardware flow control disabled (RTS and CTS signals)
  - Receive and transmit enabled
*/

/* Enable GPIO clock */
    RCC_APB2PeriphClockCmd(USART_1_GPIO_CLK | RCC_APB2Periph_AFIO, ENABLE);
/* Enable USART1 Clock */
    RCC_APB2PeriphClockCmd(USART_1_CLK, ENABLE);
/* PCLK1 = HCLK/4 */
    RCC_PCLK1Config(RCC_HCLK_Div4);
    USART_Cmd(USART_1, DISABLE);
    USART_InitStructure.USART_BaudRate =usart_speed;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
/* Configure USART1 */
    USART_Init(USART_1, &USART_InitStructure);
/* Enable USARTy Receive and Transmit interrupts USART1*/
    USART_ITConfig(USART_1, USART_IT_RXNE, ENABLE);  //zezwolenie na przerwanie odbioru
//USART_ITConfig(USART_1, USART_IT_TXE, ENABLE);  //zezwolenie na przerwanie transmisji dopiero, gdy trzeba
będzie wysłać dane
/* Enable the USART1 */
    USART_Cmd(USART_1, ENABLE);
}
```

Listing 3. Szkielet procedury obsługi przerwania od USART1

```
/**
  * obsługa przerwania portu szeregowego USART1
  */
//-----------------------------------------------------
void USART_1_IRQHandler(void)
{
//obsługa przerwania USART1
    if(USART_GetITStatus(USART_1, USART_IT_RXNE) != RESET)
    {
    //przepisanie odebranego bajtu z rejestru USART1 do buforu odbioru
        bufor_odbiorczy =USART_ReceiveData(USART_1);
    }
    if(USART_GetITStatus(USART_1, USART_IT_TXE) != RESET)
    {
    //przepisanie kolejnego bajtu do wysłania z buforu transmisji do rejestru transmisji USART1
        if (licznik_wysyłanych_danych !=0)
        {
        //w buforze transmisji są bajty do wysłania USART-em
            USART_SendData(USART_1, bufor_nadawczy);
            licznik_wysyłanych_danych--;
        }
        else
        {
        //cała zawartość buforu kołowego została wysłana, wyłączenie zezwolenia na przerwanie transmisji
            USART_ITConfig(USART_1, USART_IT_TXE, DISABLE);
        }
    }
}
```

Listing 4. Konfigurowanie NVIC do zgłaszania przerwania przy odbiorze bajtu z USART

```
void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
/* Configure the NVIC Preemption Priority Bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
/* Enable the USART1 Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = USART_1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

Listing 5. Deklarowanie buforów kołowych i zmiennych do obsługi USART

```
char bufor_rx_USART1[ROZMIAR_BUF_RX_TX];  //bufor odbiorczy USARTA1
char bufor_tx_USART1[ROZMIAR_BUF_RX_TX];  //bufor transmisji USART1
//wskaźnik do początku bufora odbiorczego USART1
char *p_bufor_rx_USART1;
//bajt określający całkowity rozmiar bufora odbiorczego USART1
int  bufor_rx_USART1_rozmiar;
/* wskaźnik bieżącej pozycji w buf.odbiorczym dla danych zapisywanych w buf. Odpowiednik *p_WE */
char *p_rx_USART1_in;
/* wskaźnik bieżącej pozycji w buf.odbiorczym dla danych odczytywanych z buf. Odpowiednik *p_WY */
char *p_rx_USART1_out;
char *p_bufor_tx_USART1;  //wskaźnik do początku buforu transmisji USART1
int  bufor_tx_USART1_rozmiar;  //bajt określający całkowity rozmiar bufora transmisji UART1
/* wskaźnik bieżącej pozycji w buf.nadawczym dla danych zapisywanych w buf. Odpowiednik *p_WE */
char *p_tx_USART1_in;
/* wskaźnik bieżącej pozycji w buf.nadawczym dla danych odczytywanych z buf. Odpowiednik *p_WY */
char *p_tx_USART1_out;
```

```
USART1_Inicjacja(9600);  //port USART
inicjowany do pracy z szybkością 9600
```

Procedury obsługi przerwań nadawania i odbioru należy standardowo umieścić w pliku *stm32f10x_it.c* przeznaczonym do implementowania procedur obsługi wszystkich przerwań kontrolera. Szkielet procedury może wyglądać tak, jak na **listingu 3**.

Wszystkie przerwania USART1 obsługiwane są przez tą samą procedurę *USART_1_IRQHandler*. Dla identyfikacji źródła przerwania bada się ustawienia flag procedurą *USART_GetITStatus*.

Jeżeli jest ustawiona flaga *USART_IT_RXNE* oznacza to, że w rejestrze odbiorczym USARTA znajduje się odebrany nowy bajt. Zawartość rejestru odczytywana jest do zmiennej programu *bufor_odbiorczy*. Po zakończeniu przerwania program w pętli głównej będzie mógł się dalej zająć daną umieszczoną w buforze. Z powodów praktycznych bufor powinien być raczej wielobajtową tablicą a nie zmienną jak na przykładowym listingu.

Jeżeli jest ustawiona flaga *USART_IT_TXE* to oznacza, że zostało zakończone wysyłanie kolejnego bajtu za pomocą USART i rejestr nadawczy jest pusty. Zazwyczaj wysyła się więcej niż 1 bajt i należy przewidzieć zastosowanie w programie zmiennej *licznik_wysylanych_danych*. Jeżeli stan zmiennej jest większy od zera z obszaru pamięci *bufor_nadawczy* należy pobrać kolejny bajt do wysłania i przy pomocy funkcji *USART_Send_Data()* umieścić go w rejestrze nadawania USART. Jeżeli nie ma już danych do wysłania powinno wyłączyć się przerwanie nadawania. W końcu należy włączyć kontroler kontrolera przerwań NVIC (**listing 4**).

Od tej chwili każdy odebrany bajt będzie wywoływał przerwanie w wyniku, którego zostanie umieszczony w buforze odbiorczym. Jeżeli zajdzie konieczność wysłania jakiejś liczby danych z buforu transmisji należy tylko włączyć przerwanie transmisji

```
USART_ITConfig(USART1, USART_IT_TXE,
ENABLE);  // enable TX interrupt
```

Listing 6. Przykładowa procedura obsługi przerwania od USART

```
/* obsługa przerwania portu szeregowego USART1
void USART_1_IRQHandler(void)
{
//obsługa przerwania USART1
    if(USART_GetITStatus(USART_1, USART_IT_RXNE) != RESET)
    {
    //przepisanie odebranego bajtu z rejestru USART1 do buforu kołowego
        *p_rx_USART1_in =USART_ReceiveData(USART_1);
        p_rx_USART1_in++;
        if (p_rx_USART1_in >=(p_bufor_rx_USART1 +bufor_rx_USART1_rozmiar))
        {
        //przewinięcie wskaźnika na początek
            p_rx_USART1_in =p_bufor_rx_USART1;
        }
    }
    if(USART_GetITStatus(USART_1, USART_IT_TXE) != RESET)
    {
    //przepisanie kolejnego bajtu do wysłania z bufora kołowego do rejestru USART1
        if (p_tx_USART1_in !=p_tx_USART1_out)
        {
        /w buforze transmisji są znaki do wysłania uart-em
            USART_SendData(USART_1, *p_tx_USART1_out);
            p_tx_USART1_out++;
            if (p_tx_USART1_out >=(p_bufor_tx_USART1 +bufor_tx_USART1_rozmiar))
            {
                p_tx_USART1_out =p_bufor_tx_USART1;
            }
        }
        else
        {
        //cała zawartość buforu kołowego została wysłana, wyłączenie zezwolenia na przerwanie
transmisji
            USART_ITConfig(USART_1, USART_IT_TXE, DISABLE);// disable tx interrupt
        }
    }
}
```

Na koniec kilka słów o ulepszeniu sposobu buforowania danych odbieranych i wysyłanych. Dobrym rozwiązaniem może być zastosowanie buforów kołowych.

Bufory kołowe

Zazwyczaj korzystając z UART przesyła się więcej niż 1 bajt. Dla uwolnienia programu głównego od konieczności ciągłej kontroli stanu rejestrów odbiorczego i nadawczego UART korzysta się z przerwań i buforów. Dane do wysłania zapisywane są w buforze nadawczym, z którego automatycznie pobierane są przez podprogram przerwania. Z kolei dane odbierane przy pomocy przerwania automatycznie zapisywane są w buforze odbiorczym. Jeżeli rozmiar buforów jest odpowiednio duży program główny może zająć się przetwarzaniem zgromadzonych w nich danych w czasie pomiędzy obsługą innych zadań.

Buforem może być obszar pamięci zadeklarowany jako tablica. Kolejne dane będą zapisywane i odczytywane do tablicy przy pomocy zmiennej indeksującej. W wypadku takiego bufora programista musi zadbać, aby wartość kolejnego indeksu do odczytu a zwłaszcza do zapisu nie przekroczył maksymalnego rozmiaru tablicy. Zapis poza zadeklarowaną tablicą może doprowadzić do trudnych do przewidzenia efektów z załamaniem się programu włącznie.

Bufor kołowy jest sztuczką programistyczną pozwalającą uniknąć takiego niebezpieczeństwa. Zasadę działania bufora kołowego pokazano na **rysunku 2**.

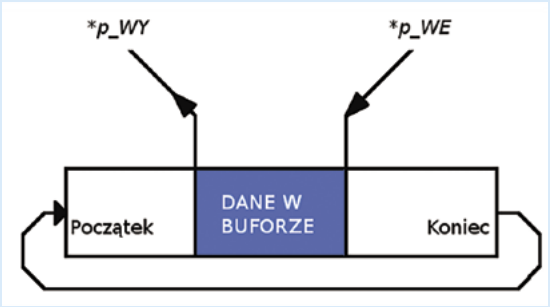
Tak jak w przypadku normalnej tablicy dane do bufora są zapisywane i odczytywane począwszy od początku do końca. Najważniejsza różnica polega na tym, że po osiągnięciu końca zapis nowych bajtów danych kontynuowany jest od jego początku. W przypadku odczytu z bufora sytuacja jest identyczna. Przy niewielkiej komplikacji konstrukcji programu unika się niebezpieczeństwa wyjścia poza zadeklarowany obszar. Niebezpieczeństwo nadpisania starych nieodczytanych danych nowymi jest łatwiejsze do opanowania i zazwyczaj nie grozi „załamaniem" programu.

Do obsługi bufora najwygodniej używać wskaźników. Za określenie pozycji danych do zapisu odpowiada wskaźnik **p_WE*, dane do odczytu wskazuje wskaźnik **p_WY*. Przed pierwszym użyciem bufora należy oba wskaźniki zainicjować wskazaniem na jego początek. Zapis kolejnych danych powoduje przesuwanie wskaźnika **p_WE* , podczas odczytu danych przechowywanych w buforze to samo dzieje się ze wskaźnikiem **p_WY*. Ilość danych aktualnie znajdujących się w buforze to różnica pomiędzy pozycjami *p_WE – p_WY*. Gdy wskaźniki wskazują na ten sam obszar bufora oznacza to, że jest on pusty.

Sytuacja nieco się skomplikuje, gdy po przewinięciu wskaźnik **p_WE* znajdzie się na początku bufora a wskaźnik odczytu **p_WY* pod jego koniec. W takim przypadku dla obliczenia ilości danych w buforze należy zsumować różnicę pomiędzy pozycją końca bufora i wskaźnikiem **p_WY* oraz różnicę pomiędzy wskaźnikiem **p_WE* a pozycją początku bufora.

Dla interfejsu USART można posłużyć się dwoma buforami kołowymi przeznaczonymi dla nadawania i odbioru. Procedury związane z przerwaniami będą musiały ulec modyfikacji. Jednak najpierw należy zadeklarować kilka pomocniczych zmiennych w tym wskaźniki **p_WE* i **p_WY* bufora odbiorczego i nadawczego. Może to wyglądać, jak na **listingu 5**.

Przed użyciem buforów niektóre zmienne należy zainicjować:



Rysunek 2. Zasada działania bufora kołowego

```
p_bufor_rx_USART1 = &bufor_rx_USART1[0];
bufor_rx_USART1_rozmiar =
sizeof(bufor_rx_USART1);
p_rx_USART1_in = &bufor_rx_USART1[0];
p_rx_USART1_out = &bufor_rx_USART1[0];
p_bufor_tx_USART1 = &bufor_tx_USART1[0];
bufor_tx_USART1_rozmiar =
sizeof(bufor_tx_USART1);
p_tx_USART1_in = &bufor_tx_USART1[0];
p_tx_USART1_out = &bufor_tx_USART1[0];
```

Przykładową procedurę obsługi przerwania USART korzystającą z buforów kołowych pokazano na **listingu 6**.

Program demonstracyjny

Przygotowany program PanEduSTM32F_Demo1_UART zawiera wszystkie opisane wcześniej elementy. Pakiet plików przeznaczony jest dla kompilatora KEIL. Jak zawsze pliki należy umieścić w podkatalogu *\Project\STM32F10x_StdPeriph_Template* zainstalowanej biblioteki *STM32F10x Standard Peripherals Firmware Library*.

Program umożliwia wypisanie za pomocą klawiatury Panelu Edukacyjnego swojego tekstu na wyświetlaczu. Klawiatura działa podobnie do klawiatury telefonu komórkowego. Klawisz S11-Esc pozwala skasować ostatni znak i cofnąć kursor. Po naciśnięciu S12-Entr tekst z wyświetlacza wysyłany jest poprzez konwerter USB Panelu i gniazdo J4 do komputera. Z kolei tekst przesyłany z komputera jest wyświetlany na wyświetlaczu Panelu począwszy od pozycji kursora. W przypadku tekstu dłuższego niż 32 znaki kolejne bajty danych nadpisywane są na wcześniejszych. Program nie obsługuje kodów polskich liter.

UART jest zaprogramowany do pracy z następującymi ustawieniami: 9600 bps, 8 bitów danych, 1 bit stopu, bez bitu parzystości i sprzętowej kontroli przepływu. Praca w trybie asynchronicznym. Na Panelu Edukacyjnym należy założyć zwory na złączach: JP5, JP2, JP1.

Jeżeli po podłączeniu do portu USB komputera Panel nie zostanie automatycznie wykryty będzie to oznaczać, że należy ręcznie zainstalować sterowniki konwertera USB FT230X. Sterowniki można pobrać ze strony producenta *http://www.ftdichip.com/*.

Do testów na komputerze należy uruchomić dowolny program terminala. Ja użyłem Br@ya v.1.93b. Ponieważ nie najlepiej sobie radził z wysokim numerem portu USB pod jakim automatycznie się zainstalowały sterowniki, zmieniłem ręcznie numer na niższy od 10. W Windows7 robi się to po otwarciu *Control Panel* → *Hardware and Sound* → *Device Manager* → *Ports (COM, LPT)*. Następnie należy kliknąć na *USB Serial Port* z numerem, pod którym został zarejestrowany Panel Edukacyjny. Po wybraniu *Port Settings* → *Advanced* należy ręcznie wskazać nowy numer portu i zatwierdzić wybór.

Ryszard Szymaniak, EP