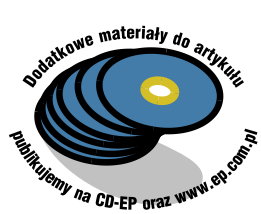
KaRTOS

8–bitowe jądro czasu rzeczywistego, część 3



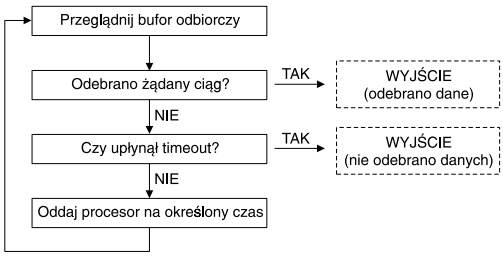
W poprzedniej części cyklu zaprezentowano pierwszą aplikację demonstracyjną dla systemu KaRTOS. Składała się z dwóch prostych zadań migających LED–ami oraz trzeciego, które korzystając z systemowego modułu obsługującego port szeregowy wyświetlało na terminalu ekran powitalny „Hello World tu KaRTOS!". Teraz dokończymy prezentację wspomnianego modułu oraz poznamy kolejny odpowiedzialny za odmierzanie czasu rzeczywistego. Na zakończenie zaimplementujemy aplikację zegara, który wyświetla na ekranie terminala aktualną datę i godzinę.

KaRTOS_UART – systemowy moduł obsługi portu szeregowego

Można zaryzykować twierdzenie, że najpopularniejszym (jak dotąd) interfejsem wykorzystywanym do obsługi przewodowej komunikacji szeregowej jest doskonale znany wszystkim Czytelnikom uniwersalny asynchroniczny odbiornik–nadajnik (UART). Umożliwia on prowadzenie komunikacji z wykorzystaniem popularnych standardów RS232, RS485 lub RS422. Właściwie każdy mikrokontroler jest obecnie wyposażany w interfejs UART, dlatego też jego obsługa przez system jest niezbędna.

Konfiguracji obsługi portu szeregowego w systemie KaRTOS dokonujemy ustawiając odpowiednie wartości zmiennych kompilatora zawartych w pliku *KaRTOSUart.h*, który znajduje się w katalogu *\KaRTOS\ATMega8*. Interesującą nas aktualnie część wspomnianego pliku przedstawiono na **list. 6**. Przystąpimy teraz do omówienia znaczenia poszczególnych parametrów. Wartości zmiennych: `UART_ROZM_BUF_OUT` oraz `UART_ROZM_BUF_IN` wyznaczają rozmiary buforów UART–a, odpowiednio: nadawczego i odbiorczego.

W jaki sposób określamy wielkości buforów i od czego one zależą? Prościej jest udzielić jednoznacznej odpowiedzi na drugą część pytania, ponieważ ograniczeniem systemowym jest wielkość bufora nie przekraczająca 255 bajtów. Istnieje jednak jeszcze ograniczenie związane z rozmiarem pamięci RAM dostępnej w mikrokontrolerze. Z pewnością pozbawione sensu jest (poza wyjątkowymi sytuacjami) zadeklarowanie buforów nadawczego i odbiorczego o rozmiarze 250 bajtów każdy, mając do dyspozycji kontroler ATmega8 (posiada 1 kB pamięci RAM). Niemal połowa dostępnej pamięci RAM zostałaby w tym przypadku wykorzystana przez moduł obsługi portu szeregowego. W jaki sposób zatem optymalnie wybrać rozmiar bufora? Aby odpowiedzieć na to pytanie należałoby poznać sposób działania modułu KaRTOS_UART w przypadku odbierania i wysyłania danych.



Rys. 12. Algorytm działania funkcji KaRTOSUartGet

Odbieranie danych z portu szeregowego w systemie KaRTOS

Wywołanie funkcji pozwalającej odebrać ciąg bajtów z portu szeregowego ma postać: `KaRTOSUartGet(u08 u08EndBajt, u08 u08IleBajtowEnd, u16 u16timeout)`. Przyjmuje ona następujące parametry:

- `u08 u08EndBajt` – zmienna zawierająca znak kończący odbieranie ciągu,
- `u08 u08IleBajtowEnd` – zmienna określająca po ilu wystąpieniach w ciągu odbieranych danych ustalonego bajtu należy zakończyć odbieranie danych,
- `u16 u16timeout` – szesnastobitowa zmienna określająca maksymalny czas w milisekundach, przez jaki funkcja będzie oczekiwać na odebranie żądanego ciągu.

Na pierwszy rzut oka funkcja przedstawiona powyżej może wydawać się zbyt skomplikowana i zagmatwana, ale jak zaraz pokażemy na przykładzie, jest bardzo elastyczna i użyteczna. Załóżmy, że chcemy odebrać z portu dwie dane, po których występuje znak nowej linii (naciśnięcie <entera> na klawiaturze powoduje wysłanie dwóch znaków – „powrót karetki" ('\r') i „nowa linia" ('\n')). Ciąg na który oczekujemy ma więc postać np.: *„123'\r'4567'\r'"*. A tak będzie wyglądało wywołanie wyżej przedstawionej funkcji, która zrealizuje postawione zadanie:

```
KaRTOSUartGet('\r',2,1000)
```

czyli odbierz ciąg znaków, aż do wystąpienia drugiego znaku '\r', lecz oczekuj nie dłużej niż 1 sekundę (1000 ms). Funkcja w odpowiedzi zwraca liczbę odebranych znaków (w naszym przykładzie 9). Prześledź-

List. 6. Parametry konfiguracji obsługi portu szeregowego zawarte w pliku KaRTOSUart.h

```
#define UART_ROZM_BUF_OUT          10
#define UART_ROZM_BUF_IN           5
#define UART_OKRES_WYSYLANIA_MS    10
#define UART_OKRES_ODBIERANIA_MS   10
```

Autor zachęca Czytelników do kształtowania treści kolejnych odcinków cyklu. Napisz w mailu czy bardziej interesuje Cię teoria działania czy raczej wolisz aby prezentowane były przykładowe aplikacje i projekty dla systemu KaRTOS. Inne uwagi również mile widziane.

my teraz co dzieje się po wywołaniu powyższej funkcji (algorytm przedstawiono na **rys. 12**):

1. Przeglądany jest cały bufor odbiorczy. Jeśli znajduje się w nim żądana liczba (*u08IleBajtowEnd*) znaków końca ciągu (*u08EndBajt*), funkcja zakończy swoje działanie zwracając liczbę odebranych bajtów.
2. Sprawdzany jest czas, który upłynął już w oczekiwaniu na dane. Jeśli przekroczył on zadaną wartość (*u16 u16timeout*), funkcja kończy działanie zwracając wartość 0 – nie odebrano żądanego ciągu znaków.
3. Funkcja oddaje procesor innemu zadaniu na określony w milisekundach zmienną *UART_OKRES_ODBIERANIA_MS* czas, po czym rozpoczyna działanie określone w punkcie 1.

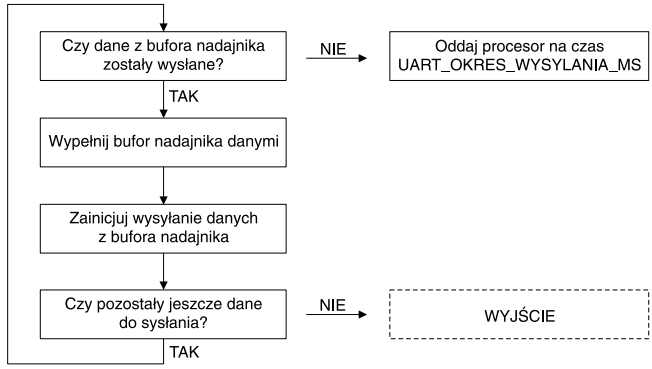
Po przedstawieniu i przeanalizowaniu powyższego przykładu możemy teraz określić, jaki powinien być minimalny rozmiar bufora odbiorczego z punku widzenia naszej aplikacji. Zwróćmy uwagę, że gdyby był on mniejszy niż 9 bajtów, to nigdy nie odebralibyśmy powyżej przedstawionego ciągu znaków. Wniosek: wielkość bufora odbiorczego nie może być mniejsza od wielkości maksymalnego ciągu danych, jaki chcemy jednorazowo odebrać. Pozostało jeszcze pochylić się nad wspomnianym powyżej parametrem *UART_OKRES_ODBIERANIA_MS*. Określa on, jak często następuje przeglądanie bufora odbiorczego w poszukiwaniu znaków kończących odbieranie danych. Oczywiście im częściej przeglądamy bufor, tym szybciej zauważymy nadejście oczekiwanych danych, lecz również mocniej obciążamy procesor. Ustalenie odpowiedniej wartości jest więc kompromisem pomiędzy szybkością reakcji po odebraniu oczekiwanych danych, a mocą obliczeniową, którą możemy poświęcić bez zaburzenia pracy pozostałych zadań. Oczywistym jest, że przeglądanie bufora co 1 milisekundę nie ma sensu jeśli dane przesyłane są z prędkością poniżej 9600 b/s, gdyż w tym przypadku przesłanie jednego bajtu wraz z bitem startu i stopu trwa ponad 1 milisekundę.

Na **list. 7** przedstawiono ciąg instrukcji umożliwiających odbieranie z portu szeregowego ciągu danych zakończonych znakiem '#'. Najpierw następuje inicjalizacja portu i konfiguracja do pracy z prędkością 38,4 kb/s.

List. 7. Ciąg instrukcji odbierających dane zakończone znakiem '#'

```
KaRTOSUartInit(12,0);     //-38,4 kbps dla zegara 8,00 MHz
for(;;)         //-pozostań w nieskończonej pętli
{
KaRTOSUartOpen();       //-otwórz port
KaRTOSUartStartRec();     //-uruchom odbiornik
i = KaRTOSUartGet('#',1,1000); //-odbierz dane
KaRTOSUartClose();        //-zamknij port
};
```

Następnie w nieskończonej pętli dokonujemy: otwarcia portu (funkcja *KaRTOSUartOpen*), uruchomienia odbiornika funkcją *KaRTOSUartStartRec* i oczekujemy na nadejście danych przez 1 sekundę. Zamykamy port, a w zmiennej i znajduje się liczba bajtów odebranych z portu.



Rys. 13. Algorytm wysyłania danych poprzez port szeregowy w systemie KaRTOS

Wysyłanie danych do portu szeregowego w systemie KaRTOS

Sposób wysyłania danych przez port szeregowy pobieżnie przedstawiono w poprzedniej części artykułu. Teraz skupimy się na przeanalizowaniu algorytmu wysyłającego dane.

Działanie funkcji wysyłającej dane prześledźmy na przykładzie. Załóżmy, że w buforze *buf[30]* znajduje się 25 bajtów danych, które chcemy wysłać przez port szeregowy. Wywołujemy zatem funkcję *KaRTOSUartSend(buf,25,SEND_RAM)*, która wyśle nasze 25 bajtów z bufora znajdującego się w pamięci RAM zgodnie z algorytmem przedstawionym na **rys. 13**:

1. Sprawdzenie czy zakończyła się poprzednia transmisja danych (wszystkie dane z bufora nadawczego zostały wysłane). Jeśli nie, następuje okres oczekiwania o długości zadeklarowanej zmienną kompilatora *UART_OKRES_WYSYLANIA_MS*,
2. Do bufora nadawczego ładowane są kolejne bajty ze wskazanego podczas wywołania funkcji bufora (w naszym przykładzie buf), aż do momentu wypełnienia go w całości lub załadowania wszystkich danych przeznaczonych do wysłania,
3. Inicjacja procesu wysyłania danych z bufora nadawczego do portu USART mikrokontrolera. Wysyłanie to realizowane jest za pomocą przerwań,
4. Sprawdzenie czy pozostały jeszcze dane do wysłania nie przepisane do bufora nadawczego. Jeśli nie, to funkcja kończy swoje działanie. Jeśli natomiast w buforze buf pozostały jeszcze dane, które należy wysłać, wracamy do punktu 1.

Analizując przedstawiony powyżej algorytm można zauważyć relacje panujące pomiędzy rozmiarem bufora nadawczego określonego zmienną kompilatora *UART_ROZM_BUF_OUT* a ilością danych przeznaczonych do wysłania. Możliwe są dwie sytuacje:

- liczba bajtów do wysłania jest mniejsza lub równa rozmiarowi systemowego bufora nadawczego. W tej sytuacji po skopiowaniu danych do wysłania ze wskazanego bufora do bufora nadawczego i inicjacji procesu wysyłania funkcja kończy swoje działanie,
- liczba bajtów do wysłania jest większa niż rozmiar systemowego bufora nadawczego. W tym przypadku dane przeznaczone do wysłania nie zmieszczą się w buforze nadawczym. Funkcja wysyłająca skopiuje zatem ich część (do zapełnienia bufora nadawczego) i zainicjuje wysyłanie. Następnie co jakiś czas (określony w zmiennej kompilatora *UART_OKRES_WYSYLANIA_MS*) będzie sprawdzać czy skopiowane uprzednio dane zostały już wysłane. Jeśli tak, nastąpi ładowanie i wysyłanie następnej partii danych. Proces ten zakończy się po skopiowaniu ostatniej ich części do bufora nadawczego.

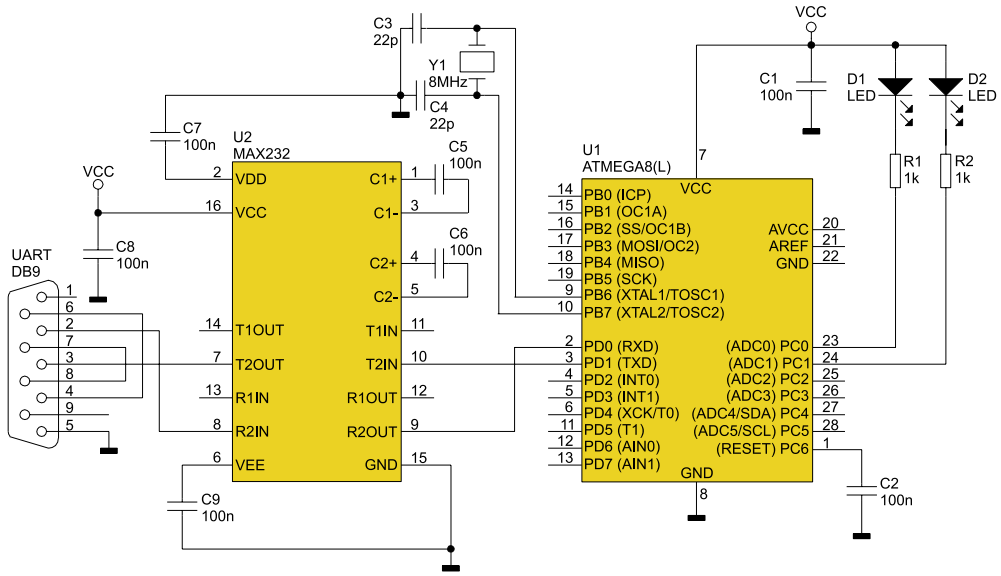
Wtedy po zainicjowaniu wysyłania funkcja zakończy swoje działanie.

Z przedstawionych powyżej informacji na temat sposobu obsługi wysyłania danych realizowanym w systemie KaRTOS łatwo można podjąć decyzję o tym jakie wartości przypisać zmiennym: *UART_ROZM_BUF_OUT* i *UART_OKRES_WYSYLANIA_MS*. Proponujemy zastosowanie następujących kryteriów: rozmiar bufora nadawczego (*UART_ROZM_BUF_OUT*) niech będzie równy rozmiarowi średniej wielkości ciągu danych, które zamierzamy wysyłać. Jeśli najdłuższym ciągiem w naszej aplikacji jest przykładowo „*Witamy w programie demonstracyjnym*" (rozmiar: 34 bajty), a najkrótszym „*Podaj liczbę*" (rozmiar: 12 bajtów), rozmiar bufora wyznaczamy na: (34+12)/2=23 bajty. W większości jednak aplikacji projektowanych na małe mikrokontrolery ograniczeniem wielkości bufora nadawczego jest niewielki rozmiar dostępnej pamięci RAM. Nadmienić należy, że jedyną konsekwencją zmniejszenia rozmiaru bufora nadawczego jest wydłużenie czasu potrzebnego na wysłanie danych o rozmiarze przekraczającym rozmiar tego bufora. Zmienna *UART_OKRES_WYSYLANIA_MS* powinna mieć wartość nie mniejszą niż iloczyn czasu potrzebnego na wysłanie jednego bajtu i rozmiaru bufora nadawczego. Dla przykładu wysyłamy dane ośmiobitowe z jednym bitem startu i jednym bitem stopu (w sumie 10 bitów) z prędkością 38400 b/s. Skoro w sekundzie w powyższych warunkach możemy wysłać maksymalnie 3840 bajtów (38400 b/s/10b), zatem czas potrzebny na wysłanie jednego jest równy 1/3840 sekundy=0,27 ms. Jeśli rozmiar bufora nadawczego ustaliliśmy przykładowo na 10 bajtów to *UART_OKRES_WYSYLANIA_MS* nie powinien być mniejszy niż 3 ms (10x0,27 ms = 2,7 ms).

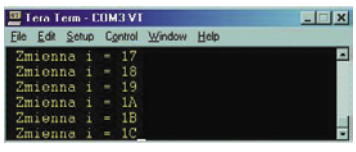
Na **list. 8** przedstawiono kod wysyłający dane przez port szeregowy. Najpierw inicjalizujemy kontroler USART znaną już funkcją, a następnie co dwie sekundy wysyłamy w formacie BCD wartość zmiennej *i* inkrementowanej w każdym obiegu pętli. Wynik działania powyższej mini aplikacji przedstawia zrzut ekranowy na **rys. 14**.

List. 8. Ciąg instrukcji wysyłających dane poprzez port szeregowy

```
u08 i;
KaRTOSUartInit(12,0);     //-38400 bps dla 8,0 MHz
for(;;)
{
KaRTOSUartOpen(); //-otwórz port
KaRTOS_UART_PRINT(„\r\n Zmienna i = „); //-wypisz
KaRTOSUartSendByteBCD(i); //-wyślij wartość i
KaRTOSUartClose();   //-zamknij port
i++;          //-inkrementuj wartosc i
TimeSleepms(2000);   //-zaczekaj 2 sekundy
};
```



Rys. 15. Schemat ideowy układu na którym uruchomiono aplikację KaRTOS CLOCK



Rys. 14. Wynik działania kodu z list. 6

Systemowe funkcje KaRTOS-a do obsługi portu szeregowego

Aby móc efektywnie wykorzystać moduł obsługi portu szeregowego, niezbędne jest poznanie funkcji umożliwiających odbieranie i wysyłanie danych w różnorodnym formacie i na różne sposoby. Wszystkie funkcje dostępne w wersji 3.01 systemu wraz z opisem ich działania i przyjmowanymi zmiennymi zebrano w **tab. 3**.

Aplikacja demonstracyjna – KaRTOS_CLOCK

Napiszemy teraz aplikację zegara–kalendarza, który będzie wyświetlał w oknie terminala aktualną datę i godzinę z dokładnością jednej milisekundy. Ustawianie zegara odbywać się będzie przez wprowadzenie z klawiatury aktualnej daty i czasu. Nasza aplikacja będzie działać na mikrokontrolerze, który był już przez nas wykorzystywany – ATmega8(L). Schemat elektryczny zegara pokazano na **rys. 15**. Na **list. 9** przedstawiono główną pętlę aplikacji pobierającej czas systemowy i wysyłającej go do portu szeregowego. Funkcja *TimeGetSystime* jest częścią modułu KaRTOSTime i pozwala pobrać czas systemowy oraz datę (sam czas lub samą datę) i zapisać go we wskazanym buforze. Wywołanie funkcji ma postać:

*TimeGetSystime(u08 *ptr, u08 opcja)*, gdzie:

- *u08 *ptr* – jest wskaźnikiem do bufora, w którym zastanie zapisana data i/lub czas systemowy,
- *u08 opcja* – jeśli ma wartość równą 0, do bufora zostanie pobrana tylko data (3 bajty) w formacie dzień/miesiąc/rok; jeśli ma wartość 1, do bufora trafi data i czas (8 bajtów) w formacie godzina/minuta/sekunda/milisekunda starszy bajt/milisekunda młodszy bajt/dzień/miesiąc/rok; jeśli natomiast ma inną wartość, pobrany zostanie tylko czas (5 bajtów) w formacie godzina/minuta/sekunda/milisekunda starszy bajt/milisekunda młodszy bajt.

W prezentowanej aplikacji wywołujemy funkcję z parametrem „opcja" o wartości 1 pobierając datę i czas.

Tab. 3. Funkcje obsługi UART dostępne w wersji 3.01 systemu

Lp.	Nazwa funkcji	Opis działania i parametrów
1.	Void KaRTOSUartInit (u16 u16Baudrate,u08 u08doubleSpeed)	Ustawia porty mikrokontrolera wykorzystywane przez USART, inicjalizuje sterownik USART. Funkcja przyjmuje: u16Baudrate – wartość ładowana do rejestru UBRRL/UBRRH określająca prędkość transmisji danych, u08doubleSpeed – jeśli równa 1 włącza, jeśli różna od 1 wyłącza podwojenie prędkości transmisji,
2.	void KaRTOSUartOpen(void)	Otwiera port szeregowy do transmisji danych (wysyłania/odbioru).
3.	void KaRTOSUartClose(void)	Zamyka port szeregowy po zakończeniu transmisji danych.
4.	void KaRTOSUartSend (u08* pu08Bufor,u08 u08liczba,u08 opcja)	Wysyła ciąg danych z określonej lokalizacji w pamięci ROM lub RAM. Funkcja przyjmuje: *pu08Bufor – wskaźnik do bufora zawierającego dane do wysłania, u08liczba – liczba danych w bajtach do wysłania ze wskazanego bufora, opcja – jeśli równa 0 – dane wysyłane z pamięci programu (ROM), jeśli 1 – z pamięci RAM (SEN) SEND_PROG–0; SEND_RAM–1
5.	void KaRTOSUartSendByteASCII (u08 u08Bajt)	Wysyła bajt przez port szeregowy. Funkcja przyjmuje: u08Bajt – bajt do wysłania.
6.	void KaRTOSUartSendByteBCD (u08 u08Bajt)	Wysyła bajt przez port szeregowy w formacie BCD. Funkcja przyjmuje: u08Bajt – bajt do wysłania.
7.	void KaRTOSUartSendDEC (u16 u16Liczba,u08 u08poz)	Wysyła liczbę szesnastobitową przez port szeregowy w formacie dziesiętnym. u16Liczba – liczba do wysłania. u08poz – zmienna określająca minimalną liczbę pozycji wyświetlania. (np. jeśli równe 3, to: u16Liczba = 4 –> wysłane zostanie: „004”; u16Liczba = 16 –> wysłane zostanie: „016”; u16Liczba = 125 –> wysłane zostanie: „125”; u16Liczba = 1895 –> wysłane zostanie: „1895”
8.	void KaRTOSUartStartRec(void)	Rozpoczyna nasłuchiwanie portu szeregowego. Dane pojawiające się na magistrali przed wywołaniem tej funkcji nie zostaną zapisane w systemowym buforze odbiorczym.
9.	u08 KaRTOSUartGet (u08 u08EndBajt, u08 u08IleBajtowEnd, u16 u16timeout)	Odbiera z portu żądany ciąg bajtów. Funkcja zwraca liczbę odebranych bajtów pomniejszoną o jeden. Funkcja przyjmuje: u08EndBajt – wartość bajtu będącego flagą końca transmisji, u08IleBajtowEnd – liczba wystąpień flag końca transmisji powodujących zakończenie odbierania danych, u16timeout – maksymalny czas oczekiwania na nadejście żądanych danych (w milisekundach).
10.	u08 KaRTOSUartGetChar(u16 u16timeout)	Odbiera z portu jeden bajt. Funkcja zwraca kod ASCII odebranego znaku lub zero jeśli upłynął timeout. Funkcja przyjmuje: u16timeout – maksymalny czas oczekiwania na nadejście bajtu (w milisekundach).

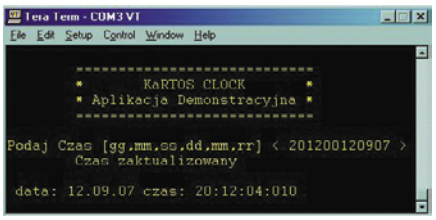
Warto tutaj zwrócić uwagę, iż bufor, do którego zostaną zapisane pobierane przez funkcję dane nie może być mniejszy niż 8 bajtów. W przeciwnym wypadku nastąpi nadpisanie przypadkowych obszarów pamięci. Popełnione błędy tego typu są jednymi z najtrudniejszych do wykrycia. W dalszej części programu używamy dobrze już znanych funkcji wysyłając w zgrabnej wizualnie formie dane z bufora do portu szeregowego podłączonego do komputera PC. Po zamknięciu portu realizujemy oczekiwanie o tak dobranym czasie trwania, aby przebieg całej pętli zajmował dokładnie 1 sekundę, co skutkuje regularnym odświeżaniem informacji na ekranie terminala. Zrzut ekranowy obrazujący działanie aplikacji KaRTOS CLOCK przedstawiono na **rys. 16**. Pozostało jeszcze zaimplementowanie funkcji, która pozwoli ustawić czas naszego zegara. Funkcję tę przedstawiono na **list. 10**. Pobrane z portu szeregowego dwanaście znaków interpretowane są kolejno jako godzina (0…23), minuta (0…59), sekunda (0…59), dzień (1…31), miesiąc (1…12) i rok (0…99). Na rozpoczęcie wprowadzania powyższych danych mamy 20 sekund, a po upływie tego czasu funkcja zwróci wartość 0xFE i zegar wystartuje z domyślnymi ustawieniami, tj. o północy pierwszego stycznia 2000 roku (zrzut z **rys. 17**). Jeśli podczas wprowadzania danych popełnimy błąd (którakolwiek z wprowadzonych danych przekroczy dopuszczalny zakres), funkcja zwróci pozycję w buforze na której wystąpił błąd (zrzut z **rys. 18** – dzień miesią-

List. 9. Główna pętla aplikacji KaRTOS CLOCK

```
for(;;)
{
TimeGetSystime(u08Bufor,1);      //-pobierz czas systemowy
//-prezentacja czasu i daty na terminalu
KaRTOSUartOpen();
KaRTOS_UART_PRINT(„\r”);
            KaRTOS_UART_PRINT(„ data: „);
            KaRTOSUartSendDEC((u16)(u08Bufor[5]),2);
            KaRTOS_UART_PRINT(„.”);
            KaRTOSUartSendDEC((u16)(u08Bufor[6]),2);
            KaRTOS_UART_PRINT(„.”);
            KaRTOSUartSendDEC((u16)(u08Bufor[7]),2);

            KaRTOS_UART_PRINT(„ czas: „);
            KaRTOSUartSendDEC((u16)(u08Bufor[0]),2);
            KaRTOS_UART_PRINT(„:”);
            KaRTOSUartSendDEC((u16)(u08Bufor[1]),2);
            KaRTOS_UART_PRINT(„:”);
            KaRTOSUartSendDEC((u16)(u08Bufor[2]),2);
            KaRTOS_UART_PRINT(„:”);
            temp = (u16)(u08Bufor[3]);
            temp >>=8;
            temp += (u16)(u08Bufor[4]);
            KaRTOSUartSendDEC(temp,3);
        KaRTOSUartClose();

        TimeSleepms(970);
    };
```



Rys. 16. Uruchomiony KaRTOS CLOCK

Rys. 17. Timeout wprowadzania czasu

ca ma wartość 52). Jeśli wprowadzanie danych przebiegnie pomyślnie, funkcja zwróci wartość 0xFF, a zegar systemowy zostanie ustawiony za pomocą kolejnej funkcji z modułu *KaRTOSTime* o nazwie *TimeSetSystime*. Przyjmuje ona jako argument sześć bajtów oznaczających kolejno rok, miesiąc, dzień, godzinę, minutę i sekundę. Siódmy z argumentów jest szesnastobitowy i może mieć wartość w zakresie 0…999 a określa milisekundę. Po ustawieniu czasu zobaczymy działającą aplikację (rys. 16).

Konfiguracja systemu do uruchomienia aplikacji KaRTOS CLOCK

Co jeszcze pozostało do wyjaśnienia, a właściwie przypomnienia? Oczywiście sposób skonfigurowania samego systemu, aby możliwe było uruchomienie naszego najnowszego zegara. Kolejne etapy przedstawione są poniżej w porządku przyjętym podczas prezentacji poprzednio uruchamianej aplikacji zaprezentowanej w poprzedniej części cyklu.

Konfiguracja zadań

Ponieważ aplikacja zegara składa się z jednego zadania, w pliku *main.c* należy uruchomić systemowy proces bezczynności oraz *Task_1*:

```
KaRTOSTaskInit(&KaRTOSIdleTa-
sk,1,255,50);
KaRTOSTaskInit(&(Task_1),TASK_1_PI-
D,TASK_1_PRIORITY,TASK_1_STACK);
```

Konfiguracji zadania *Taski_1* dokonujemy w pliku *main.h*, na przykład w taki sposób:

```
#define TASK_1_PID              10
#define TASK_1_STACK            150
#define TASK_1_PRIORITY         10
```

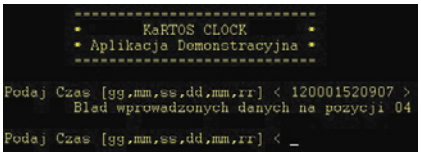
Konfiguracja systemu

Dokonujemy jej edytując plik *KaRTOS.conf*, który znajduje się w katalogu *KaRTOS\ATMega8*. Należy uruchomić następujące moduły systemu:

- KaRTOS_RTC – moduł zegara RTC odmierzającego czas,
- KaRTOS_EXT_TIME – rozszerzenie modułu zegara o datę,
- KaRTOS_UART_ON – do komunikacji,
- KaRTOS_STRING – umożliwia operowanie na ciągach danych, w nim zawarta jest np. funkcja: *KaRTOSStringASCII2DEC*.

Wartości pozostałych zmiennych jak w projekcie *Hello_world_tu_KaRTOS*:

```
#define SYS_STACK               20
#define NO_TASKS_RAM_ADDR       619
#define KaRTOS_1MS_OCR_WART     125
```



Rys. 18. Błędnie wprowadzony czas

Konfiguracja pinów mikrokontrolera

Jedyne piny kontrolera jakie wykorzystuje aplikacja KaRTOS CLOCK, to te używane do transmisji szeregowej. Są one jednak odpowiednio ustawiane przez funkcję systemową inicjalizującą UART i nie ma potrzeby edytowania pliku *KaRTOS\ATMega8\Initm8.c*. Jeśli Czytelnik chciałby wykorzystać jedną lub obie diody podłączone do PORTU C (np. jako wizualizacja upływającego czasu), należałoby ustawić odpowiednio wspomniany port. DDRC = 0x03 oraz PORTC = 0x03 – piny 0 i 1 portu są wyjściami w stanie wysokim.

Implementacja kodu zadań

Została już zaprezentowana powyżej.

Kompilacja

Przebiega standardowo. Po uruchomieniu konsoli systemu Windows (*Menu Start–>Programy–>Akcesoria–>Wiersz Poleceń*), przejściu do katalogu projektu (polecenie *cd*) i wydaniu polecenia *make* etap kompilacji mamy za sobą.

Programowanie

Za pomocą swojego ulubionego programatora ładujemy plik wynikowy kompilacji do pamięci kontrolera oraz ustawiamy odpowiednio opcje zegara taktującego. (rezonator zewnętrzny lub oscylator wewnętrzny o częstotliwości 8,0 MHz).

Gotowy projekt jest zawarty w pliku archiwum *KaRTOS CLOCK.zip*.

Mariusz Żądło
iram@poczta.onet.pl

List. 3.5. Funkcja ustawiająca zegar

```
u08  PobierzCzas(void)
{
    u08  i,  j,  wsk;

    KaRTOSUartOpen();
        KaRTOS_UART_PRINT(„\rPodaj  Czas  [gg,mm,ss,dd,mm,rr]  <  „);

        for(wsk=0;  wsk<12;  wsk++)
        {
            KaRTOSUartStartRec();
            i  =  KaRTOSUartGetChar(20000);  //-timeout  20sek
            if(i  !=  0)
            {
                u08Bufor[wsk]  =  i;
                KaRTOSUartSendByteASCII(i);
            }else
            {
                KaRTOS_UART_PRINT("  brak  danych  >");
                KaRTOSUartClose();
                return(0xFE);
            }
        }

        KaRTOS_UART_PRINT("  >");
    KaRTOSUartClose();

    i  =0;
    j  =  KaRTOSStringASCII2DEC(u08Bufor[i],u08Bufor[i+1]);     //-godziny
    if(j  >  24)  return(i);
    u08Bufor[0]  =  j;
    i  +=2;
    j  =  KaRTOSStringASCII2DEC(u08Bufor[i],u08Bufor[i+1]);     //-minuty
    if(j  >  59)  return(i);
    u08Bufor[1]  =  j;
    i  +=2;
    j  =  KaRTOSStringASCII2DEC(u08Bufor[i],u08Bufor[i+1]);     //-sekundy
    if(j  >  59)  return(i);
    u08Bufor[2]  =  j;
    i  +=2;
    j  =  KaRTOSStringASCII2DEC(u08Bufor[i],u08Bufor[i+1]);     //-dzień
    if(j  >  31)  return(i);
    if(j  ==  0)  return(i);
    u08Bufor[3]  =  j;
    i  +=2;
    j  =  KaRTOSStringASCII2DEC(u08Bufor[i],u08Bufor[i+1]);     //-miesiac
    if(j  >  12)  return(i);
    if(j  ==  0)  return(i);
    u08Bufor[4]  =  j;
    i  +=2;
    j  =  KaRTOSStringASCII2DEC(u08Bufor[i],u08Bufor[i+1]);     //-rok
    u08Bufor[5]  =  j;

    TimeSetSystime(u08Bufor[5],u08Bufor[4],u08Bufor[3],u08Bufor[0],u08Bufor[
  1],u08Bufor[2],0);

    return(0xFF);
}
```