STM32 dla początkujących (i nie tylko)

RTC, czyli zegar i kalendarz

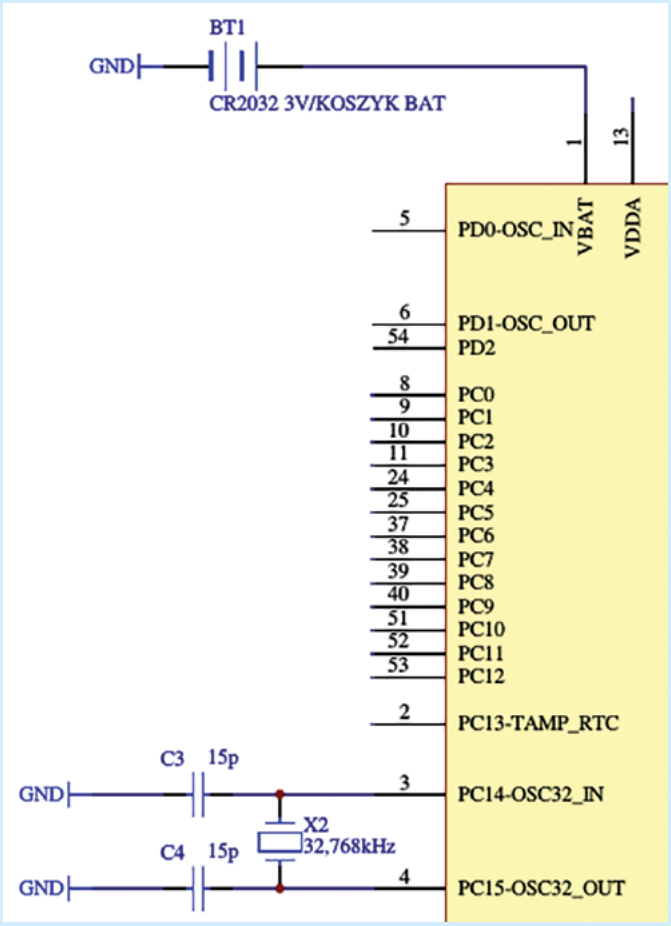
Układ zegara RTC stanowi integralną część kontrolerów STM32. Podane informacje i przykłady powinny pomóc w jego zastosowaniu w różnych aplikacjach wymagających odmierzania czasu.

W artykule omówiono wybrane zastosowania modułu RTC zaimplementowanego w STM32. Opisano funkcje biblioteczne służące do obsługi RTC oraz Backup Domain. Zaprezentowano przykładowe procedury obsługi służące do odczytu czasu, daty, korekty daty. Na koniec zaprezentowano program demonstracyjny, w którym praktycznie zrealizowano przykładowy zegar.

RTC zastosowania

Najbardziej oczywistym sposobem wykorzystania zegara RTC (*Real-time clock*) jest budowa różnego typu czasomierzy z kalendarzem i budzikiem, jednak zastosowań może być znacznie więcej. Dołączenie do kontrolera pamięci wymiennej typu karta SD i zapis danych w formacie standardowych plików wymaga podania rzeczywistego czasu i daty ich utworzenia. Można oczywiście markować te wartości, ale znacznie rozsądniej jest użyć odczytanego z zegara realnego czasu. Za pomocą RTC można też określać i zapamiętywać moment wystąpienia rejestrowanych zdarzeń. W różnego typu sterownikach zegar można wykorzystać do inicjowania akcji w zależności od pory dnia czy daty, do pomiaru czasu itd. Zegar RTC może być użyty do wybudzenia urządzenia zasilanego z baterii z trybu obniżonego poboru mocy.

W dalszej części tekstu skupimy się na podstawowych sposobach wykorzystania RTC do pomiaru czasu. Z kwestiami precyzyjnego strojenia zegara czy jego nietypowego użycia np. do wybudzania systemu można się zapoznać chociażby w notach aplikacyjnych dostępnych na stronach www.st.com powiązanych z konkretnym typem mikrokontrolera.



Rysunek 1. Elementy zewnętrzne niezbędne do pracy RTC

RTC zasada działania i budowa

Układ RTC do działania potrzebuje źródła precyzyjnych impulsów zegarowych. W mikrokontrolerze STM32 można wykorzystać w tym celu np. wewnętrzne impulsy taktujące, jednak lepiej posłużyć się specjalnym oscylatorem, którego działanie jest podtrzymywane za pomocą dodatkowej baterii litowej. Na **rysunku 1** pokazano dodatkowe elementy zewnętrzne potrzebne do pracy RTC w takiej właśnie konfiguracji. Standardowo stosuje się zewnętrzny oscylator kwarcowy pracujący z częstotliwością 32768 Hz (2^{15} Hz). Bateria litowa o napięciu 3 V bateria podtrzymuje działanie zegara RTC w czasie, gdy zasilanie główne mikrokontrolera jest wyłączone. Gdy napięcie zasilania zostanie załączone, to następuje automatyczne przełączenia RTC z baterii na zasilanie mikrokontrolera. Po przełączeniu pobór prądu z baterii jest minimalny, co przedłuża jej żywotność.

Na rysunku 2 pokazano schemat blokowy zegara RTC stanowiącego integralną część kontrolera. Głównym elementem jest 32 bitowy rejestr *RTC_CNT*. Rejestr zlicza precyzyjnie odmierzone impulsy czasowe o częstotliwości 1 Hz, czyli o okresie 1 s. Impulsy formowane są w bloku podzielnika *RTC prescaler* z impulsów zewnętrznego rezonatora kwarcowego 32768 Hz.

Rejestr *RTC_CNT* współdziała z rejestrem *RTC_ALR*. Gdy zawartość obydwu jest równa generowany zostaje sygnał alarmu *RTC_Alarm*. Sygnał alarmu może być źródłem przerwania lub służyć do wybudzenia kontrolera z trybu uśpienia oczywiście o ile jest w tym czasie podłączony do zasilania. Innymi źródłami przerwań mogą być sygnały sekund z podzielnika *RTC prescaler* i sygnał *RTC_Overflow* generowany w momencie przepełnienia rejestru *RTC_CNT*.

Jak to wcześniej napisałem główne bloki zegara są nieprzerwanie zasilane albo napięciem zasilania mikrokontrolera, albo podtrzymywane bateryjnie. Na rys. 2 te bloki zaznaczono ciemniejszym kolorem. Z pozostałymi układami kontrolera zegar RTC komunikuje się poprzez wewnętrzną magistralę APB1. Dokładny opis budowy i omówienie rejestrów układu RTC mikrokontrolera STM32F103 można znaleźć w dokumencie *Reference manual (CD00171190.pdf)* w sekcji *Real-time clock (RTC)*.

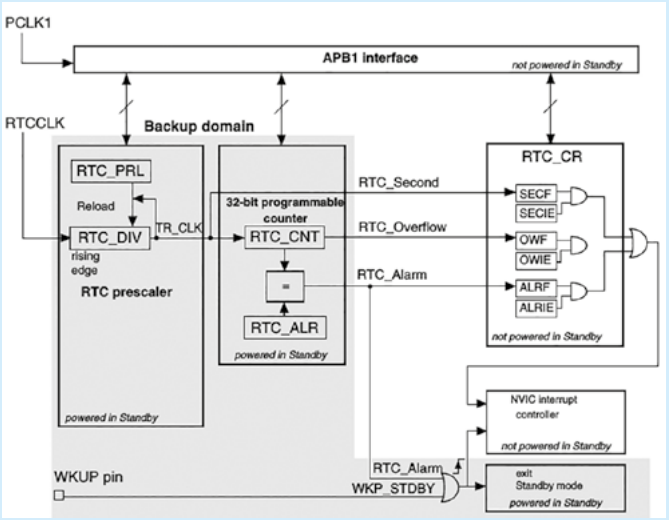
RTC i Backup Domain

Obwody zegara nie są jedynymi, których zawartość jest podtrzymywana bateryjnie w czasie odłączenia napięcia zasilania od kontrolera. W STM32F103 jest wydzielony blok dziesięciu 16-bitowych rejestrów, które razem z układami RTC tworzą tzw. *Backup Domain*. Są to rejestry ogólnego przeznaczenia i można je wykorzystać do przechowywania najważniejszych danych, które nie powinny być utracone, gdy mikrokontroler nie jest zasilany. Niektóre z tych dziesięciu rejestrów będą wykorzystane przez przykładowe oprogramowanie zegara z kalendarzem.

Ponieważ zawartość rejestrów z *Backup Domain* musi być chroniona przed przypadkowym zapisem np. w czasie przełączania napięć zasilania, dostęp do ich zawartości jest nieco bardziej skomplikowany niż do zwykłych rejestrów pamięci RAM kontrolera. Przede wszystkim, po restarcie należy zainicjować dostęp do RTC i rejestrów w *Backup Domain*. Najwygodniej zrobić to posługując się funkcjami dostarczanymi przez bibliotekę *STM32F10x Standard Peripherals Firmware Library*.

Funkcje biblioteczne

Przejrzenie funkcji *STM32F10x Standard Peripherals Firmware Library* związanych z zegarem czasu rzeczywistego jest możliwe po otworzeniu pliku *stm32f10x_stdperiph_lib_um.chm* i wpisaniu w polu wyszukiwarki skrótu RTC. Po wybraniu opcji *STM32F10x Standard Peripherals Library:Footer* i potem *RTC_Exported_Functions* zostanie wyświetlona lista dostępnych funkcji.



Rysunek 2. Schemat blokowy zegara RTC

Na szczególna uwagę zasługują *RTC_GetCounter()* i *RTC_SetCounter()* pozwalające na odczyt i zapis rejestru zegara *RTC_CNT*. Funkcja *RTC_SetAlarm()* pozwala zapisać nową wartość do rejestru alarmu *RTC_Alarm*. Istotne są także dwie ostatnie funkcje na liście: *RTC_WaitForLastTask()* – czuwająca nad prawidłowym zakończeniem kolejnej operacji związanej z dostępem do rejestrów RTC oraz *RTC_WaitForSynchro()* -synchronizująca przepływ danych pomiędzy blokami RTC a magistralą APB1.

W podobny sposób można podejrzeć funkcje związane z rejestrami z *Backup Domain*. W pasku wyszukiwarki należy wpisać BKP. Z wyświetlonej listy najbardziej interesujące są funkcje *BKP_ReadBackupRegister()* i *BKP_WriteBackupRegister()*. Dzięki nim jest ułatwiony odczyt i zapis 16-bitowej danej do wybranego rejestru z *Backup Domain*.

Inicjowanie RTC

W czasie wyłączenia głównego zasilania zegar RTC jest podtrzymywany przez baterię jednak po restarcie należy każdorazowo zainicjować dostęp do jego rejestrów.

Listing 1. Inicjowanie zegara czasu rzeczywistego

```
//inicjowanie zegara czasu rzeczywistego
void RTC_Inicjacja(void)
{
    /* Enable PWR and BKP clocks */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR | RCC_APB1Periph_BKP, ENABLE);
    /* Allow access to BKP Domain */
    PWR_BackupAccessCmd(ENABLE);
    /* Reset Backup Domain */
    //BKP_DeInit();
    /* Enable LSE 32.768 kHz external oscillator*/
    RCC_LSEConfig(RCC_LSE_ON);
    /* Wait till LSE is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
    {}

    /* Select LSE as RTC Clock Source */
    RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
    /* Enable RTC Clock */
    RCC_RTCCLKCmd(ENABLE);
    /* Wait for RTC registers synchronization */
    RTC_WaitForSynchro();
    /* Wait until last write operation on RTC registers has finished */
    RTC_WaitForLastTask();
    /* Enable the RTC Second */
    //RTC_ITConfig(RTC_IT_SEC, ENABLE);
    /* Enable the RTC Alarm */
    RTC_ITConfig(RTC_IT_ALR, ENABLE);
    /* Wait until last write operation on RTC registers has finished */
    RTC_WaitForLastTask();
    /* Set RTC prescaler: set RTC period to 1sec */
    RTC_SetPrescaler(32767); /* RTC period = RTCCLK/RTC_PR = (32.768 KHz)/(32767+1) */
    /* Wait until last write operation on RTC registers has finished */
    RTC_WaitForLastTask();
}
```

Inicjacja jest szczególnie potrzebna po każdorazowym odłączeniu i dołączeniu baterii. Jeżeli w tym czasie główne zasilanie kontrolera także było odłączone wszystkie rejestry w *Backup Domain* zostaną wyzerowane, a oscylator kwarcowy 32768 Hz zatrzymany. Procedura inicjowania może wyglądać, jak na **listingu 1**.

Najpierw do *Backup Domain* jest dołączany wewnętrzny zegar i uruchamiana komenda zezwalająca na dostęp do rejestrów. Dołączany i uruchamiany jest zewnętrzny oscylator 32768 Hz i oprogramowanie oczekuje aż jego działanie się ustabilizuje. Następnie ustawiany jest preskaler tak aby na jego wyjściu pojawiał się sygnał o okresie 1s. Dodatkowo w procedurze inicjacji może być ustawiane zezwolenie na przerwanie wywoływane przez alarm RTC. Zezwolenie na przerwanie wywoływane przez impulsy sekund *RTC_ITConfig(RTC_IT_SEC, ENABLE)* jest wyłączone (odpowiednia linia listingu jest opatrzona komentarzem). Po zakończeniu procedury inicjacji zegar RTC działa, możliwy jest także dostęp do jego rejestrów i rejestrów w *Backup Domain*.

Oprogramowanie przykładowe

Wykorzystanie wewnętrznego zegara RTC do stworzenia zegara z kalendarzem i alarmem wymaga napisania trochę dodatkowego własnego oprogramowania. Przykład *PanEduSTM32F_Demo1_RTC* pokazuje jak można to zrobić wykorzystując Panel Edukacyjny. Możliwości programu demonstracyjnego są następujące:

- Wyświetlanie czasu: godzin, minut sekund i daty: dnia, miesiąca, 2 ostatnich cyfr roku.
- *Ręczne ustawianie i korygowanie czasu i daty*.
- Programowanie alarmu w trybie 24-godzinnym.

Do prawidłowego działania program wymaga:

- *Włożenia baterii 3* V do gniazda na Panelu Edukacyjnym.
- Założenie zworek JP5 i JP3.
- *Włożenie wyświetlacza LCD 2*×16 znaków do gniazda J5.

Do obsługi jest używanych 5 przycisków klawiatury multipleksowanej Panelu Edukacyjnego:

- S1 i S2 służą do przesuwu kursora w poziomie. Kończenie niektórych funkcji wymaga wyjścia kursora poza wyświetlacz.
- S5 i S8 służą do przesuwu kursora w pionie. Kończenie niektórych funkcji wymaga wyjścia kursora poza wyświetlacz.
- S12 służy do inicjowania funkcji ustawiania parametrów zegara oraz kalendarza i akceptowania trybu alarmu.

Oprogramowanie zostało zaadaptowane z opublikowanego wcześniej w EP projektu zegara-minutnika. Ponieważ chodziło o zademonstrowanie sposobów użycia zegara RTC niektóre funkcje wcześniejszego projektu zostały usunięte, a niepotrzebne fragmenty kodu opatrzone komentarzem. Obsługę modułu zegara RTC uproszczono dla zwrócenia uwagi na najważniejsze funkcje.

Podstawowe procedury związane z dostępem do zegara RTC znajdują się w plikach *Procedury_RTC.c*

Listing 2. Odczytanie czasu z zegara RTC

```
//odczyt czasu z zegara RTC
void RTC_Odczyt_czasu(char *p_buf)
{
    uint32_t TimeVar;
    TimeVar=RTC_GetCounter();
    TimeVar=TimeVar % 86400;
    s_TimeStructVar.HourHigh=(uint8_t)(TimeVar/3600)/10;
    s_TimeStructVar.HourLow=(uint8_t)(TimeVar/3600)%10;
    s_TimeStructVar.MinHigh=(uint8_t)((TimeVar%3600)/60)/10;
    s_TimeStructVar.MinLow=(uint8_t)((TimeVar%3600)/60)%10;
    s_TimeStructVar.SecHigh=(uint8_t)((TimeVar%3600)%60)/10;
    s_TimeStructVar.SecLow=(uint8_t)((TimeVar %3600)%60)%10;
    snprintf(p_buf, 12, „%d%d:%d%d:%d%d”, s_TimeStructVar.HourHigh,
             s_TimeStructVar.HourLow, s_TimeStructVar.MinHigh,
             s_TimeStructVar.MinLow, s_TimeStructVar.SecHigh,
             s_TimeStructVar.SecLow);
}
```

Listing 3. Inicjowanie kalendarza

```
//inicjacja kalendarza
void Kalendarz_Inicjacja(void)
{
    uint16_t skompresowana_data;
    //odczyt z rejestru backup skompresowanej daty dla kalendarza
    skompresowana_data =BKP_ReadBackupRegister(BKP_REJESTR_KALENDARZA);
    //rozwinięcie daty i zapis do struktury
    s_DateStructVar.Year = (skompresowana_data >>KOMPRESJA_KALENDARZ_POZ_ROKU) & KOMPRESJA_KALENDARZ_MASKA_
ROKU;
    s_DateStructVar.Month = (skompresowana_data >>KOMPRESJA_KALENDARZ_POZ_MIESIAC) & KOMPRESJA_KALENDARZ_
MASKA_MIESIAC;
    s_DateStructVar.Day = (skompresowana_data >>KOMPRESJA_KALENDARZ_POZ_DZIEN) & KOMPRESJA_KALENDARZ_MASKA_
DZIEN;
    if (s_DateStructVar.Month ==0) s_DateStructVar.Month =1;
    if (s_DateStructVar.Day ==0) s_DateStructVar.Day =1;
}
```

Listing 4. Korygowanie daty

```
//sprawdzenie czy zegar czasu odliczył więcej niż 1 dzień, jeśli tak korekta daty
void CheckForDaysElapsed(void)
{
    uint32_t DaysElapsed;
    if((RTC_GetCounter() / SECONDS_IN_DAY) != 0)
    {
        for(DaysElapsed = 0; DaysElapsed < (RTC_GetCounter() / SECONDS_IN_DAY); DaysElapsed++)
        {
            DateUpdate();
        }
        RTC_SetCounter(RTC_GetCounter() % SECONDS_IN_DAY);
    }
}
```

i *Procedury_RTC.h*. W plikach *Funkcje_Zegara.c*, *Funkcje_Alarmu.c* i *Funkcje_Alarmu_Odliczanie.c* umieszczono kod odpowiedzialny za działanie zegara, kalendarza i alarmu. Procedury z tych plików odwołują się do podstawowych procedur z pliku *Procedury_RTC.c*.

Dla łatwiejszego zrozumienia zasady działania przykładowego programu najpierw zajmiemy się procedurami podstawowymi. Najpierw odczytem bieżącego czasu.

Odczyt bieżącego czasu z RTC

Procedura *RTC_Odczyt_czasu()* odczytuje zawartość licznika *RTC_CNT* i zamienia ją na ilość sekund, minut i godzin, które upłynęły od początku doby (**listing 2**). Procedura używa struktury *s_TimeStructVar* do przechowywania skonwertowanej liczby sekund odczytanej z rejestru *RTC_CNT* na liczbę dziesiątek i jednostek godzin, minut i sekund, które upłynęły od początku doby, czyli godziny 00:00:00. Dodatkowo, w buforze wskazywanym przez **p_buf* jest umieszczany łańcuch z informacją o odczytanym czasie przygotowany do wyświetlenia na wyświetlaczu LCD.

Odczyt bieżącej daty

Odczyt bieżącej daty realizuje procedura *Kalendarz_Inicjacja()*. Bieżąca data pamiętana jest w jednym z rejestrów *Backup Domain* czyli *BKP_REJESTR_KALENDARZA*. Dla zaoszczędzenia miejsca numery bieżącego roku, miesiąca i dnia są zapisane na kolejnych bitach i rozwijane przez procedurę zamieszczoną na **listingu 3**. Do przechowywania odczytanej daty jest wykorzystywana struktura *s_DateStructVar*. Po odczycie inna procedura *RTC_Odczyt_Daty()* formatuje odczytaną datę w łańcuch przygotowany

Listing 5. Aktualizowanie daty po upływie dnia

```
//po upływie dnia aktualizacja daty
void DateUpdate(void)
{
    Kalendarz_Inicjacja();
    if(s_DateStructVar.Month == 1 || s_DateStructVar.Month == 3 || \
       s_DateStructVar.Month == 5 || s_DateStructVar.Month == 7 ||\
       s_DateStructVar.Month == 8 || s_DateStructVar.Month == 10 \
       || s_DateStructVar.Month == 12)
    {
        if(s_DateStructVar.Day < 31)
        {
            s_DateStructVar.Day++;
        }
        /* Date structure member: s_DateStructVar.Day = 31 */
        else
        {
            if(s_DateStructVar.Month != 12)
            {
                s_DateStructVar.Month++;
                s_DateStructVar.Day = 1;
            }
            /* Date structure member: s_DateStructVar.Day = 31 & s_DateStructVar.Month =12 */
            else
            {
                s_DateStructVar.Month = 1;
                s_DateStructVar.Day = 1;
                s_DateStructVar.Year++;
            }
        }
    }
    else if(s_DateStructVar.Month == 4 || s_DateStructVar.Month == 6 \
            || s_DateStructVar.Month == 9 ||s_DateStructVar.Month == 11)
    {
        if(s_DateStructVar.Day < 30)
        {
            s_DateStructVar.Day++;
        }
    /* Date structure member: s_DateStructVar.Day = 30 */
        else
        {
            s_DateStructVar.Month++;
            s_DateStructVar.Day = 1;
        }
    }
    else if(s_DateStructVar.Month == 2)
    {
        if(s_DateStructVar.Day < 28)
        {
            s_DateStructVar.Day++;
        }
        else if(s_DateStructVar.Day == 28)
        {
        /* Leap Year Correction */
            if(CheckLeap(s_DateStructVar.Year))
            {
                s_DateStructVar.Day++;
            }
            else
            {
                s_DateStructVar.Month++;
                s_DateStructVar.Day = 1;
            }
        }
        else if(s_DateStructVar.Day == 29)
        {
            s_DateStructVar.Month++;
            s_DateStructVar.Day = 1;
        }
    }
    Kalendarz_kompresja();
}
```

do wyświetlenia na wyświetlaczu LCD. Funkcję odwrotną wykonuje procedura *Kalendarz_kompresja()*, która zwija zapisaną w strukturze datę do rozmiarów 16-bitowego rejestru i zapisuje w *BKP_REJESTR_KALENDARZA*.

Aktualizowanie daty

Po upływie doby następuje aktualizacja kalendarza, zajmuje się tym procedura *CheckForDaysElapsed()*. Pokazano ją na **listingu 4**.

Najpierw sprawdza ona czy wartość rejestru *RTC_CNT* jest większa niż *SECONDS_IN_DAY* (86400). Jest to liczba sekund odliczanych w czasie 24 godzin. Jeżeli odczytana wartość przekracza *SECONDS_IN_DAY* w pętli jest wywoływana procedura korekty daty, czyli doliczenia kolejnego dnia. Na zakończenie, do *RTC_CNT* jest wpisywana liczba sekund, która upłynęła od początku bieżącej doby. Procedura prawidłowo skoryguje datę zarówno wywołana o północy jak i o dowolnej porze dnia nawet po kilkudniowym wyłączeniu głównego zasilania kontrolera.

Korekta daty

Ze względu na skomplikowaną strukturę kalendarza (różna liczba dni zależnie od miesiąca a nawet roku) aktualizacja daty nie może po prostu doliczać kolejnego dnia, ale musi uwzględniać wspomniane komplikacje. Dla tego doliczaniem zajmuje się wyspecjalizowana procedura *DateUpdate()*. Pokazano ją na **listingu 5**. Zależnie od sytuacji procesdura:

- Dolicza kolejny dzień miesiąca.
- Zmienia numer miesiąca na kolejny.
- Zmienia numer roku na kolejny.
- Dodatkowo, jest wywoływana procedura pomocnicza *CheckLeap()* umożliwiająca wykrycie roku przestępnego i prawidłową korektę daty w lutym.

Program demo: wszystko razem

Po zerowaniu program najpierw inicjuje poszczególne układy w tym zegar RTC. Następnie sprawdza, w jakim stanie znajdował się w momencie wyłączenia zasilania: normalnego wyświetlania czasu i daty czy odliczania czasu pozostałego do alarmu. Zależnie od tego następuje wyświetlenie winiety powitalnej albo natychmiastowe przejście do wyświetlania czasu pozostałego do alarmu. Następnie następuje działanie programu w trybie automatu stanu. Zależnie od bieżącego stanu procedura *Automat_Stanu_Urzadzenia()* wywołuje odpowiednie procedury.

Jeżeli zegar znajduje się w trybie wyświetlania czasu i daty, to co pewien czas wywoływane są procedury aktualizacji czasu i daty. Jeżeli zegar znajduje się w trybie zaprogramowanego alarmu zamiast bieżącej daty i godziny wyświetlany jest czas pozostały do alarmu. Po wystąpieniu alarmu następuje powrót do normalnego wyświetlania czasu i daty.

Ryszard Szymaniak, EP