Układ RTC DS3231

Skonstruowanie i budowa zegara są często jedną z pierwszych „poważniejszych" prac początkującego elektronika. W handlu są dostępne układy RTC różnych firm, różniące się możliwościami oraz parametrami. Często początkujący konstruktorzy wybierają układy najtańsze, które spełniają oczekiwania, ale w perspektywie czasu okazuje się, że zegar śpieszy się lub spóźnia poza akceptowane ramy. Zaczyna się szukanie przyczyn – dokładanie trymerów do oscylatora, pojemności filtrujących napięcie zasilające lub nawet zamykanie układu w pudełku styropianowym w celu ograniczenia wpływu temperatury. Niestety, na dłuższą metę te czynności nie przynoszą spodziewanych efektów. Sprawa wygląda jeszcze gorzej, gdy układ RTC jest narażony na zmiany temperatury w szerokim zakresie.

Przyczyną niedokładnej pracy zegara, o której mowa we wstępie, są parametry oscylatorów zegarkowych dołączanych do układu scalonego. Podstawowy parametr niepewności częstotliwości oscylacji jest wyrażany w jednostkach „ppm". Średnio dla większości oscylatorów ten parametr jest równy ±20 ppm. Co to dokładnie oznacza? Jeżelibyśmy chcieli wyrazić ±20 ppm w procentach, to otrzymamy ±0,002%. Można powiedzieć, że to bardzo mało, ale jeśli pomnożymy tę wartość przez liczbę sekund w miesiącu, to w skrajnym wypadku otrzymamy 51,8 s. Oczywiście, jest to wartość skrajna, ponieważ raczej nigdy nie uzyskamy odchyłki maksymalnej – tylko w „jedną stronę", chociaż może tak się zdarzyć. Nie uwzględniamy tu jeszcze zmian temperatury, które znacząco pogłębiają błąd, ponieważ każdy oscylator jest kalibrowany w określonej temperaturze – dla większości jest 25°C.

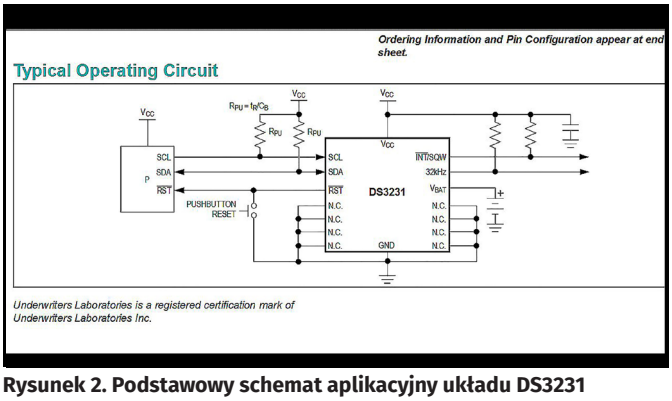
Dryft częstotliwości również jest podawany w zależności od temperatury otoczenia. Dla większości typowych oscylatorów wynosi on 0,035 ppm/°C[2]. Jeżeli zegar jest użytkowany w domu lub w pomieszczeniu o stałej temperaturze, to w zasadzie możemy nie uwzględniać dryftu, ale przecież nawet w mieszkaniu temperatura może się zmieniać o kilka stopni w ciągu doby, a dryft ma znaczenie. Jeżeli generator pracuje w zmiennych warunkach, to już nie jest tak wesoło i odchyłka o 10°C od temperatury, dla której był skalibrowany ocylator, spowoduje nam błąd rzędu 9 sekund miesięcznie. Jak temu zaradzić?

Można synchronizować wskazania zegara z wzorcem czasu, pobierając czas z serwera NTP lub korzystając z sygnału radiowego DCF. Wiąże się to jednak z dodatkowymi kosztami, skomplikowaniem programu mikrokontrolera oraz wzrostem złożoności płytki drukowanej. Można też wydać nieco więcej pieniędzy na układ scalony zegara RTC i zastosować **DS3231** firmy Maxim Integrated. Co takiego ma ten układ, czego nie mają inne?

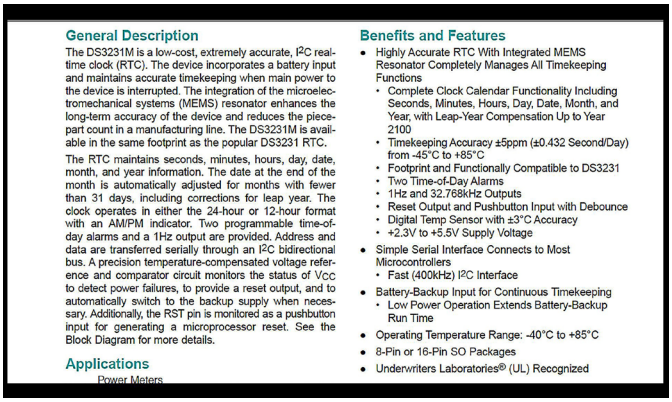
Pierwszą stronę dokumentacji układu DS3231 pokazano na **rysunku 1**. Układ obsługuje lata przestępne do 2100 roku, więc nie trzeba tego uwzględniać w programie obsługi zegara. Jednak najważniejszy parametr to pozycja mówiąca o odchyłce częstotliwości w zakresie temperatury od 0 do +40°C. Nie dosyć, że układ nie wymaga zewnętrznego oscylatora, to jeszcze ten wbudowany w jego strukturę jest kompensowany temperaturowo i w podanym zakresie odchyłka zawsze wyniesie ±2 ppm. Dodatkowo, mamy do dyspozycji rejestr o nazwie „Register for Aging Trim", za pomocą którego można dodawać lub odejmować pojemność w obwodzie generatora kwarcowego, uzyskując taki sam efekt, jak przy zastosowaniu trymera. Na uwagę zasługuje również to, że w zakresie temperatury od –40 do +85°C niepewność oscylatora wynosi jedynie ±3,5 ppm.

Układ ma wbudowany termometr, ale raczej mało dokładny. Może przydać się w roli sensora temperatury panującej w pokoju, ale raczej nie nadaje się do dokładnych pomiarów. Interfejs I²C pracuje z maksymalną częstotliwością zegarową wynoszącą 400 kHz. Układ ma wejście dla baterii podtrzymującej zasilanie zegara w razie zaniku napięcia głównego. Może przy tym pracować w dużym zakresie napięcia zasilającego – od 2,3 do 5,5 V, przy czym nie wymaga konwersji napięcia na szynie I²C. Pobór prądu w stanie aktywnym to 200 µA przy zasilaniu 3,3 V. Pobór prądu z baterii wynosi około 70 µA. Układ jest produkowany w obudowie SO16. Dostępne są również warianty w obudowie SO8 noszące oznaczenie DS3231MZ+.

Podstawowy schemat aplikacyjny układu pokazano na **rysunku 2**. Jest nieskomplikowany i myślę, że nie wymaga omawiania. Dlatego w artykule skupię się na opisie biblioteki funkcji służącej do obsługi tego ciekawego układu – jest ona dostępna w materiałach dodatkowych do tego artykułu, do pobrania z serwera FTP.



Rysunek 2. Podstawowy schemat aplikacyjny układu DS3231



General Description

The DS3231M is a low-cost, extremely accurate, I²C real-time clock (RTC). The device incorporates a battery input and maintains accurate timekeeping when main power to the device is interrupted. The integration of the microelectromechanical systems (MEMS) resonator enhances the long-term accuracy of the device and reduces the piece-part count in a manufacturing line. The DS3231M is available in the same footprint as the popular DS3231 RTC.

The RTC maintains seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator. Two programmable time-of-day alarms and a 1Hz output are provided. Address and data are transferred serially through an I²C bidirectional bus. A precision temperature-compensated voltage reference and comparator circuit monitors the status of VCC to detect power failures, to provide a reset output, and to automatically switch to the backup supply when necessary. Additionally, the RST pin is monitored as a pushbutton input for generating a microprocessor reset. See the Block Diagram for more details.

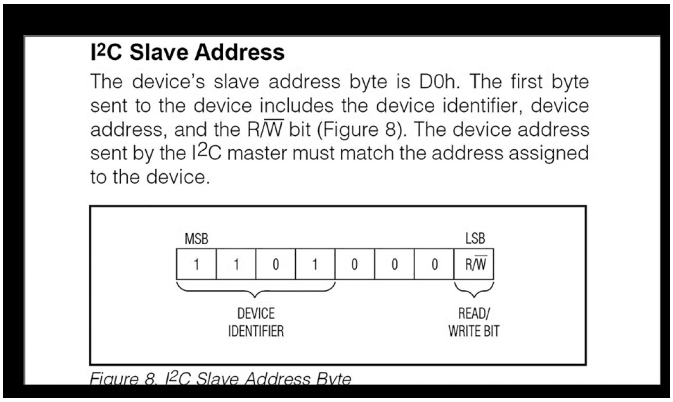
Applications

Power Meters

Benefits and Features

- Highly Accurate RTC With Integrated MEMS Resonator Completely Manages All Timekeeping Functions
  - Complete Clock Calendar Functionality Including Seconds, Minutes, Hours, Day, Date, Month, and Year, with Leap-Year Compensation Up to Year 2100
  - Timekeeping Accuracy ±5ppm (±0.432 Second/Day) from -45°C to +85°C
  - Footprint and Functionally Compatible to DS3231
  - Two Time-of-Day Alarms
  - 1Hz and 32.768kHz Outputs
  - Reset Output and Pushbutton Input with Debounce
  - Digital Temp Sensor with ±3°C Accuracy
  - +2.3V to +5.5V Supply Voltage
- Simple Serial Interface Connects to Most Microcontrollers
  - Fast (400kHz) I²C Interface
- Battery-Backup Input for Continuous Timekeeping
  - Low Power Operation Extends Battery-Backup Run Time
- Operating Temperature Range: -40°C to +85°C
- 8-Pin or 16-Pin SO Packages
- Underwriters Laboratories® (UL) Recognized

Rysunek 1. Pierwsza strona dokumentacji układu DS3231



Rysunek 3. Ustalenie adresu układu na magistrali I²C

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/24	AM/PM 20 Hours	10 Hours	Hour				Hours	1–12 + AM/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/24	AM/PM 20 Hours	10 Hours	Hour				Alarm 1 Hours	1–12 + AM/PM 00–23
0Ah	A1M4	DY/DT	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/24	AM/PM 20 Hours	10 Hours	Hour				Alarm 2 Hours	1–12 + AM/PM 00–23
0Dh	A2M4	DY/DT	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	EOSC	BBSQW	CONV	NA	NA	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32KHZ	BSY	A2F	A1F	Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	81h–7Fh
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Temperature MSB	—
12h	DATA	DATA	0	0	0	0	0	0	Temperature LSB	—

Rysunek 4. Rejestry konfiguracyjne układu DS3231

Listing 1. Zerowanie rejestru Register Control

```
void DS3231_init( void )
{
    uint8_t control = 0;
    TWI_write_buf( DS3231_ADDR, 0x0e, 1, &control );
}
```

Listing 2. Struktura do przechowywania danych czasu i daty

```
typedef union
{
    uint8_t bytes[8];
    struct
    {
        uint8_t ss;
        uint8_t mm;
        uint8_t hh;
        uint8_t dayofweek;
        uint8_t day;
        uint8_t month;
        uint8_t year;
        uint16_t next_day;
    };
} TDATETIME;
```

Adres układu na magistrali I²C jest równy **0xD0**. Pierwsze, co musimy zrobić, to zastanowić się, czy będziemy w programie wykorzystywali alarmy dostępne w układzie. Jeżeli tak, to wyprowadzenie INT/SQW nie będzie mogło dostarczać przebiegu 1 Hz.

Jeżeli zapoznamy się z zawartością rejestru o nazwie **Register Control**, to zauważymy, że po włączeniu zasilania lub restarcie domyślnie jest ustawiany bit INTCN. Oznacza to, że przerwanie od alarmu jest włączone. Aby na wyjściu INT/SQW występował przebieg 1 Hz musimy ten bit wyzerować. Ponieważ pozostałe bity też są „0”, możemy wpisać do całego rejestru „0”, jak na **listingu 1**. Jeśli to zrobimy, to na wyjściu INT/SQW będzie dostępny przebieg o częstotliwości 1 Hz, co wykorzystamy w programie jako niezależny timer i w bardzo wygodny sposób będziemy mogli w przerwaniu co 1 sekundę odczytywać aktualny czas.

Listing 4. Uwzględnienie zmiany czasu – czas zimowy/letni

```
void TIME_CORRECTION_EVENT(TDATETIME * dt)
{
    if(!time)
    {
        if((dt->ss==2)&&(dt->mm==0)&&(dt->hh==2))
        {
            if((dt->month==3)&&(dt->day>24)&&(dt->dayofweek==7))
            {
                time=1;
                DS3231_set_time(3,00,02 );
            }
        }
    }
    else
    if(time)
    {
        if((dt->ss==2)&&(dt->mm==0)&&(dt->hh==3))
        {
            if((dt->month==10)&&(dt->day>24)&&(dt->dayofweek==7))
            {
                time=0;
                DS3231_set_time(2,00,02 );
            }
        }
    }
}
```

Wykaz rejestrów układu zaczerpnięty z jego dokumentacji pokazano na **rysunku 4**. Kolejne rejestry dostępne pod adresami od 0x00 do 0x06 zawierają sekundy, minuty, godziny (do wyboru – tryb 24- lub 12-godzinny), kolejny dzień tygodnia, dzień miesiąca, miesiąc i rok, przy czym rok to tylko dwie ostatnie cyfry daty, więc jeśli chcemy podać całą liczbę roku, to początek (aktualnie liczbę „20”) musimy dodać w programie. Dalej, mamy rejestry do ustawiania dwóch alarmów, rejestry *Control*, *Status* oraz *Aging offset* do ewentualnej korekty częstotliwości oscylatora. Na samym końcu rejestry, z których możemy odczytać temperaturę struktury układu.

Nas interesuje przede wszystkim odczyt czasu i daty. Wartości te musimy zapisać do zmiennych, aby móc później zaprezentować dane dla użytkownika. Myślę, że wygodnie jest utworzyć strukturę, w której będziemy przechowywali wszystkie wartości. Pozwoli ona wygodnie odwoływać się do nich w programie, w zależności od potrzeb. Wspomnianą strukturę pokazano na **listingu 2**.

Po włączeniu zasilania zegar rozpoczyna odmierzanie czasu od „00:00:00”, więc na jego początku, na przykład w pliku „main.c”, musimy wpisać aktualny czas i datę do rejestrów układu zegara. W strukturze z listingu 2 poszczególne wartości są ułożone w kolejności rejestrów. Oprócz tego, w rozbudowanym zegarze może być przydatna informacja o kolejnym dniu roku. Uwzględnia ona rok przestępny. Z punktu widzenia tego, co napisano wcześniej, przydatna będzie funkcja, odczytująca rejestry czasu i daty i zapisująca te dane do struktury. Zawarto w niej funkcjonalność konwersji liczb binarnych na dziesiętne. Obie funkcje zamieszczono na **listingu 3**.

W strukturze z listingu 2 jest pole o nazwie *dayofweek*, która jest wyrażona wartością od 1 do 7. Jednak zwykle na wyświetlaczu

Listing 3. Obliczanie dnia roku, odczytanie rejestrów układu zegara oraz tablica z nawami dni

```
uint16_t day_of_year(TDATETIME * dt)
{
    uint16_t n,n1,n2,n3;
    n1 = floor(275 * dt->month / 9);
    n2 = floor((dt->month + 9) / 12);
    n3 = (1 + floor((dt->year - 4 * floor(dt->year / 4) + 2) / 3));
    n = n1 - (n2 * n3) + dt->day - 30;
    dt->next_day = n ;
    return dt->next_day;
}

void DS3231_get_datetime( TDATETIME * dt )
{
    uint8_t i;
    uint8_t buf[7];
    I2C_read_buf( DS3231_ADDR, 0x00, 7, buf );
    for( i=0; i<7; i++ ) dt->bytes[i] = bcd2dec( buf[i] );
}

char days[7][14] = {„poniedziałek „, „     wtorek „, „     środa „, „  czwartek „, „     piątek „, „     sobota „, „  niedziela „
};
```

pokazujemy nie liczbę, a nazwę dnia tygodnia. Konwersję najłatwiej przeprowadzić za pomocą tablicy z nazwami dni tygodnia, w której poniedziałek ma numer 1, a niedziela 7. Ponieważ występują tu polskie znaki, trzeba to obsłużyć programowo w zależności od zastosowanego wyświetlacza. Należy zwrócić uwagę, że w tablicy nazwy zaczynają się od pozycji „0”, natomiast z rejestru układu odczytujemy dni od wartości jeden, więc odwołując się w kodzie do tablicy, musimy od wartości „dayofweek” odjąć jeden.

Pozostaje nam jeszcze uwzględnić zmianę czasu na letni lub zimowy. Odpowiednią funkcję pokazano na **listingu 4**. W funkcji wykorzystano dodatkową zmienną *time*, pełniącą funkcję flagi. Dla czasu letniego zmienna przybiera wartość „1”, natomiast dla zimowego „0”. Należy to uwzględnić w momencie inicjalizacji zmiennej dla obowiązującego czasu.

Pozostaje nam już tylko na starcie programu wpisać do rejestrów układu aktualny czas i datę. Do tego celu są przeznaczone funkcje: *DS3231_set_time* dla godziny oraz *DS3231_set_date* dla daty. Obie funkcje zamieszczono na **listingu 5**. Chciałbym zwrócić uwagę, że w bibliotece nie ma funkcji do wyświetlania daty i czasu, ponieważ trudno jest przewidzieć, w jaki sposób będzie prezentowana godzina czy data w aplikacji docelowej.

Przykładową funkcję *main* pokazano na **listingu 6**. Na początku dołączamy pliki systemowe, a następnie inicjujemy interfejs I²C, ustawiając częstotliwość SCL na 400 kHz. Przy ustawianiu prędkości magistrali należy uwzględnić parametry innych dołączonych do niej układów i ewentualnie skorygować wartość *bitrate*. Teraz inicjujemy układ DS3231 oraz przerwanie od wejścia INT0. Możemy wybrać inny rejestr INT lub PCINT – wtedy należy skorygować funkcję oraz wektor przerwania. Wpisujemy do rejestrów układu aktualny czas i datę, przy czym po wpisaniu należy linie te opatrzyć komentarzem i ponownie wgrać program. Unikniemy w ten sposób błędnego odczytu po ponownym uruchomieniu programu, np. w wyniku zaniku napięcia zasilania. Nasz zegarek będzie przecież pracował na podtrzymaniu. Włączamy oczywiście zezwolenie na globalne przerwania.

Pętla główna to funkcje do wyświetlania czasu i daty (które należy wykonać samodzielnie w zależności od użytego wyświetlacza) oraz funkcja korekcji czasu letni/zimowy. Na koniec wektor przerwania, gdzie co sekundę odczytujemy aktualny czas. Życzę przyjemnej pracy z układem i tworzenia fajnych czasomierzy.

W artykule wykorzystano biblioteki do obsługi I²C z książki „Mikrokontrolery AVR język C – podstawy programowania” wydawnictwa Atnel.

Listing 5. Funkcje ustawiające czas i datę

```
void DS3231_set_time( uint8_t hh, uint8_t mm, uint8_t ss )
{
    uint8_t buf[3];
    buf[0]=dec2bcd(ss);
    buf[1]=dec2bcd(mm);
    buf[2]=dec2bcd(hh);
    TWI_write_buf( DS3231_ADDR, 0x00, 3, buf );
}

void DS3231_set_date( uint8_t year, uint8_t month, uint8_t day, uint8_t dayofweek )
{
    uint8_t buf[4];
    buf[0]=dayofweek;
    buf[1]=dec2bcd(day);
    buf[2]=dec2bcd(month);
    buf[3]=dec2bcd(year);
    TWI_write_buf( DS3231_ADDR, 0x03, 4, buf );
}
```

Listing 6. Przykładowa funkcja main

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include „I2C_TWI/i2c_twi.h”
#include „ds3231.h”

int main(void)
{
    i2cSetBitrate( 400 );
    DS3231_init();
    INT_init();
//DS3231_set_time(15,10,00 );
//DS3231_set_date(18,12,22,6);
    sei();
    while(1)
    {
        DS3231_show_time(&datetime);
        DS3231_show_date(&datetime);
        TIME_CORRECTION_EVENT(&datetime);
    }
}

ISR(INT0_vect)
{
    DS3231_get_datetime(&datetime);
}
```

Marek Rębecki
marekrebecki@wp.pl