Real-Time Clock using the Asynchronous Timer

FPSLIC™

Programmable SLI

AT94K

AT94S

Application Note

Features

- **Real-Time Clock with Very Low Power Consumption (4 µA @ 3.3V)**
- **Very Low Cost Solution**
- **Adjustable Prescaler to Adjust Precision**
- **Counts Time, Date, Month and Year with Auto Leap Year Configuration**
- **Year 2000 Compliant Date Format**
- **Can be Used on all AVR Controllers with RTC Module**
- **C Code for AT94K Inculded**

Introduction

This application note describes how to implement a Real-Time Clock (RTC) on the FPSLIC™ embedded AVR microcontroller. The implementation requires only a 32.768 kHz watch crystal. The application has very low power consumption because the microcontroller operates in Power Saving mode, most of the time. In Power Saving mode the microcontroller is sleeping with only a timer running. The timer is clocked by the external crystal, an on every timer overflow, the time, date, month and year are counted. The advantages of implementing a RTC in software compared to an external hardware RTC are obvious:

- Lower Cost
- Few External Components
- Lower Power
- Greater Flexibility

The C code with the Real-Time Clock routines can be found in the FPSLIC Software section of the Atmel web site (www.atmel.com), under the **3050.c** archive.

Theory of Operation

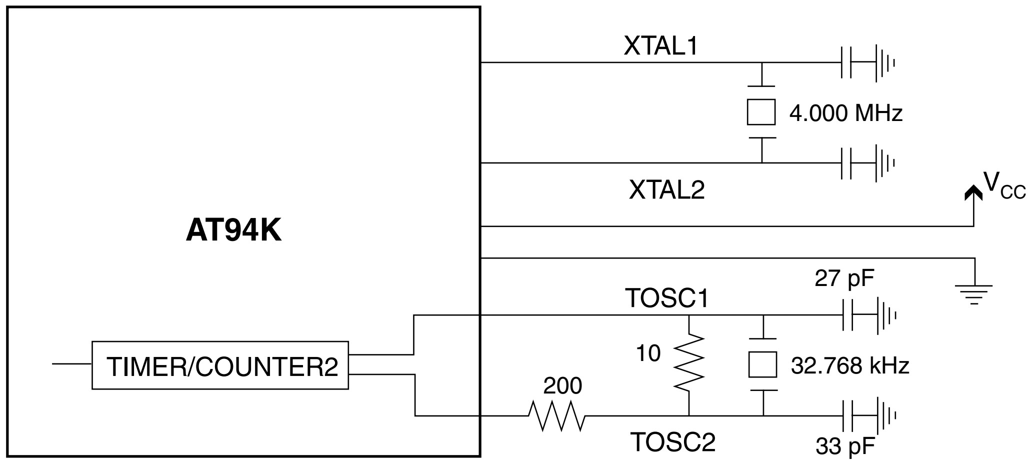
The implementation of an RTC utilizes the asynchronous operation of the RTC module. In this mode, Timer/Counter2 runs independently from the CPU clock.

The FPSLIC microcontroller operates from the 4 MHz main clock source in normal operation. When low power operation is desired, the microcontroller operates in Power Down mode, with only the asynchronous timer running from an external 32.768 kHz crystal.

The software real-time clock (RTC) is implemented using an 8-bit timer/counter with overflow interrupt. The software controls the overflow interrupt to count clock and calendar variables. The Timer Overflow interrupt is used to update the software variables “second”, “minute”, “hour”, “date”, “month” and “year” at the correct intervals.

Rev. 3050A–FPSLI–05/02

Figure 1. Oscillator connection for Real-Time Clock



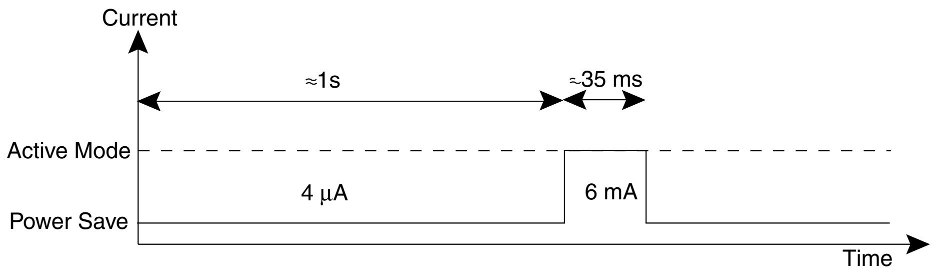
Since the amount of time for the timer/counter to complete one overflow is always the same, each timer variable will be incremented by a fixed number with every timer overflow. The timer overflow interrupt routine is used to perform this task.

To reduce power consumption, the microcontroller enters a Power Saving mode, in which all on-chip modules are disabled except for the RTC; the microcontroller typically consumes less than 4 µA in this mode, see Table 1. The device will wake up on the Timer Overflow interrupt. The updates of the timer variables are performed during the active period.

Then the microcontroller re-enters the Power Saving mode until the next timer overflow occurs. Figure 2 and Figure 3 show the operating time difference of the microcontroller in Power Saving mode versus Active mode.

The total power consumption equals to the addition of the power consumption in Power Saving mode and the power consumption in Active mode. It takes less than 100 cycles to update the timer variables in the interrupt routine, but with a 4 MHz main clock it only takes 25 µs. The power consumption for this period is neglible. The wake-up time for the controller can be programmed to 35 ms for use with the external crystal, or 1 ms for use with the ceramic resonator. Figure 2 shows an example of a circuit that wakes up once every second to update the RTC and the power consumption for the two types of clock source:

Figure 2. Current Figures for Crystal Oscillator, 35 ms Startup Time

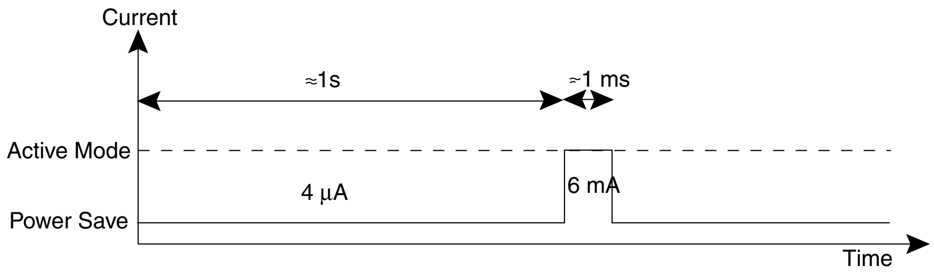


Total current consumption per second:

= (1 sec * 4 µA) + (35 ms * 6 mA) = 4 µAs + 210 µAs = 214 µAs

This shows that the dominating part of the current consumption is in Active mode.

Figure 3. Current Figures for Ceramic Resonator, 0.5 ms Startup Time



Total current consumption per second:

= (1 sec * 4 μA) + (1 ms * 6 mA) = 4 μAs + 6 μAs = 10 μAs

This shows that by reducing the startup time the current consumption is reduced from 100 μAs to 7 μAs.

Table 1. Current Consumption by the AVR Controller in Each Mode

Mode	Typical	Max
Active 4 MHz, 3.3 V_{CC}	4 mA	6.0 mA
Idle 4 MHz, 3.3 V_{CC}	1.8 mA	2.0 mA
Power Down 4 MHz, 3.3 V_{CC}	<1.0 μA	2.0 μA
Power Save 4 MHz, 3.3 V_{CC}	<4.0 μA	6.0 μA

Calculation

Given the frequency of the watch crystal, the user can determine the time for each tick in the Timer/Counter by selecting the desired prescale factor. As shown in Table 2, CS22, CS21 and CS20 in the TCCR2 (Timer/Counter2 Control Register) define the prescaling source of the Timer/Counter, where CK is the frequency of the watch crystal. For example, if CK equals 32,768 Hz, the Timer/Counter will tick at a frequency of 256 Hz with a prescaler of CK/128.

Table 2. Timer/Counter0 Prescale Select

CS22	CS21	CS20	Description	Overflow Period
0	0	0	Timer/Counter2 is stopped	–
0	0	1	CK(1)	1/64s
0	1	0	CK/8(1)	1/8s
0	1	1	CK/32(1)	1/4s
1	0	0	CK/64(1)	1/2s
1	0	1	CK/128(1)	1s
1	1	0	CK/256(1)	2s
1	1	1	CK/1024(1)	8s

Note: 1. CK = 32,768 Hz

Configuration Example

As shown in Figure 1, the crystal should be connected directly between pins TOSC1 and TOSC2. The oscillator is optimized for use with a 32,768 Hz watch crystal, or an external clock signal in the interval of 0 Hz - 256 kHz. In this example, the eight LEDs in port D are used to display the RTC. The LED on port D pin 0 will change state every second. The next 6 LEDs represent the minute in binary, and the LED on pin 7 stays on for 1 hour and off for the next.

Considerations should be taken when clocking the timer/counter from an asynchronous clock source. A 32.768 kHz crystal have a stabilization time up to 1 second after power up. The controller must therefore not enter Power Saving mode less than a second after power up. Care must be taken when changing to asynchronous operation. When updating the timer register the data is transferred to a temporary register and latched after two external clock cycles. The ASynchronous Status Register (ASSR) contains status flags that can be checked to control that the written register is updated.

Implementation

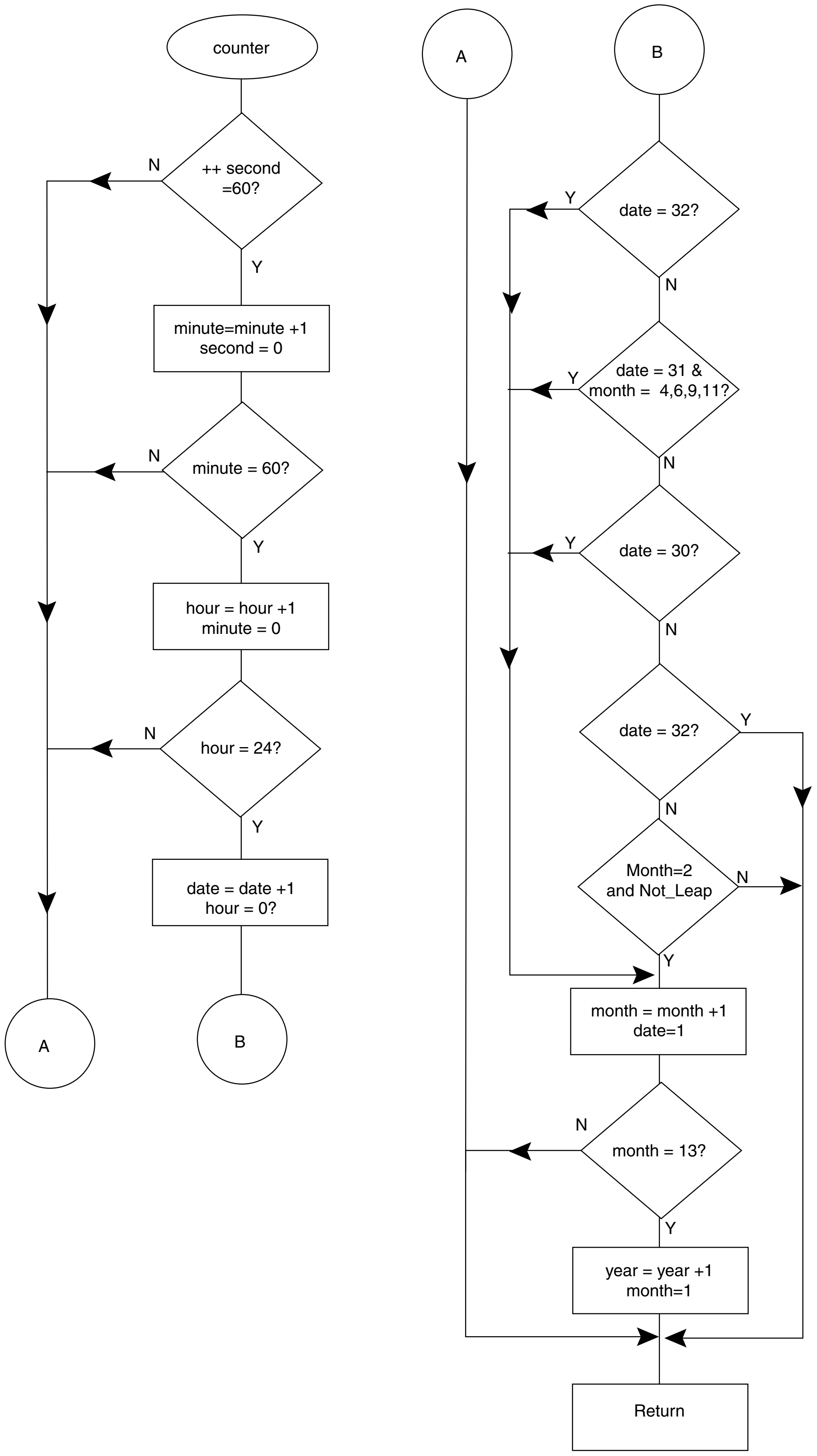
The software consists of two subroutines. “Counter” is the Timer/Counter overflow service routine, which updates all the timer variables whenever a timer overflow occurs. The other one, “not_leap”, corrects the date for leap years. The main program sets up all the necessary I/O registers to enable the RTC module and controls the power down sequence.

The AS2 bit in the ASSR is set to configure Timer/Counter2 to be clocked from an external clock source. Only this timer can perform asynchronous operations. The start value for the timer is reset and the desired prescaler value is selected. To synchronize with the external clock signal the program wait for the ASSR register to be updated. TOIE2 bit in the TIMSK (Timer/Counter Interrupt Mask Register) is then set to enable Timer/Counter2 Overflow interrupt. The Global Interrupt Enable bit in SREG (Status Register) also has to be set to enable all interrupts. SM1 and SM0 bit in MCUR (MCU Control Register) are set to select Power Saving mode. The SLEEP instruction will then place the controller in Sleeping mode. A loop in the main program executes the SLEEP instruction.

“Counter” Overflow Interrupt Routine

The interrupt routine is executed every time a timer overflow occurs. It wakes up the MCU to update the timer variables. An interrupt procedure cannot return or accept any variables. A global structure with timer variables are declared to keep track of time: “second”, “minute”, “hour”, “date”, “month” and “year”. Since the time required to complete one timer overflow is known, “second” will be incremented by a fixed number every time. Once it reaches 60, “minute” is incremented by 1 and “second” is set to 0.

Figure 4. Flow Chart, CounterInterrupt Routine



“not_leap” Subroutine

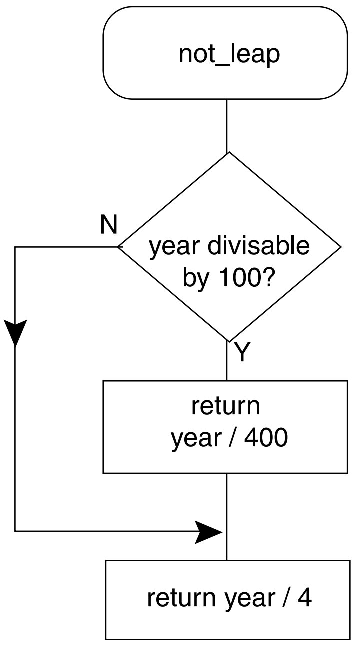
This routine checks whether or not it is a leap year. It returns true if the year is not leap and false for leap. It is considered a leap year if both of the following conditions are satisfied:

1. The year is divisible by 4, and
2. If the year is divisible by 100, it also has to be divisible by 400.

Accuracy

The RTC on the microcontroller maintains high accuracy as long as the watch crystal is accurate. Asynchronous operation allows the timer to run without any delays even when the CPU is under heavy computation. However, a small neglible discrepancy does occur because the timer variables are not updated in parallel. By the time they are finished updating, they deviate from the Timer/Counter very slightly. The largest discrepancy occurs when all the timer variables are overflowed. At this moment, “second” is 59, “minute” is 59, “hour” is 23, and so on. It takes 94 cycles for the MCU to complete the update. At a 4Mhz CPU clock, the error between the RTC and the watch crystal will not exceed 23.5 µs found by 94 / (4 * 10^6). A typical error should be 6 µs since 24 cycles are needed to update “second”. This error does not accumulate since the timer is always synchronous with the watch crystal.

Figure 5. Flow Chart



Resources

Table 3. CPU and Memory Usage

Function	Code Size (bytes)	Cycles	Example Register	Interrupt	Description
main	104	–	R16	Timer0 Overflow	Sets the necessary configuration
counter	356	–	R16, R17, R30, R31	–	Updates the variables
not_leap	48	10 (typical)	R16, R17, R20, R21	–	Checks for leap year
Total	508	–		–	–

Table 4. Peripheral Usage

Peripheral	Description	Interrupts Enabled
TOSC1, TOSC2	Connected to external crystal	–
Timer/counter2	Real-time clock	Timer/counter2 overflow
8 I/O pins on port D	Flashing LEDs (example only)	–

```
/**** A P P L I C A T I O N NOTE *************************
 *
 * Title:           Real-Time Clock
 * Version:         1.01
 * Last Updated:   03/25/02
 * Target:          AT94K
 * Description
 * This application note shows how to implement a Real-Time Clock utilizing a secondary
 * external oscillator. Included a test program that performs this function, which keeps
 * track of time, date, month, and year with auto leap-year configuration. 8 LEDs are used
 * to display the RTC. The 1st LED flashes every second, the next six represents the
 * minute, and the 8th LED represents the hour.
 *
 *************************************************************************************/

#include <ioat94k.h>
#include <ina90.h>

char        not_leap(void);

type        def struct{
unsigned    char second;                            //enter the current time, date, month, and year
unsigned    char minute;
unsigned    char hour;
unsigned    char date;
unsigned    char month;
unsigned    int year;
             }time;

 time t;

void C_task main(void)                             //C_task means "main" is never called from another function
{
    int     temp0,temp1;

    for(temp0=0;temp0<0x0040;temp0++)               // Wait for external clock crystal to stabilize
    {
        for(temp1=0;temp1<0xFFFF;temp1++);
    }
    DDRD=0xFF;
    TIMSK &=~((1<<TOIE2)|(1<<OCIE2));               //Disable TC2 interrupt
    ASSR |= (1<<ASR);                               //set Timer/Counter2 to be asynchronous from the CPU clock
                                                    //with a second external clock(32,768kHz)driving it.
    TCNT2 = 0x00;
    TCCR2 = 0x05;                                   //prescale the timer to be clock source / 128 to make it
                                                    //exactly 1 second for every overflow to occur
    while(ASSR&0x07);                               //Wait until TC2 is updated
    TIMSK |= (1<<TOIE2);                            //set 8-bit Timer/Counter2 Overflow Interrupt Enable
    _SEI();                                         //set the Global Interrupt Enable Bit
    while(1)
    {
        MCUR = 0x38;                               //entering sleeping mode: power save mode
        _SLEEP();                                  //will wake up from time overflow interrupt
        _NOP();
        TCCR2=0x05;                                // Write dummy value to Control register
        while(ASSR&0x07);                          //Wait until TC2 is updated
    }
}

interrupt [TIMER2_OVF_vect] void counter(void)     //overflow interrupt vector
{

    if (++t.second==60)                            //keep track of time, date, month, and year
    {
        t.second=0;
        if (++t.minute==60)
        {
            t.minute=0;
            if (++t.hour==24)
            {
                t.hour=0;
                if (++t.date==32)
                {
                    t.month++;
                    t.date=1;
                }
                else if (t.date==31)
                {
                    if ((t.month==4) || (t.month==6) || (t.month==9) || (t.month==11))
                    {
                        t.month++;
                        t.date=1;
                    }
                }
                else if (t.date==30)
                {
                    if(t.month==2)
                    {
                       t.month++;
                       t.date=1;
                    }
                }
                else if (t.date==29)
                {
                    if((t.month==2) && (not_leap()))
                    {
                        t.month++;
                        t.date=1;
                    }
                }
                if (t.month==13)
                {
                    t.month=1;
                    t.year++;
                }
            }
        }
    }
    PORTD=~(((t.second&0x01)|t.minute<<1)|t.hour<<7);


}

char not_leap(void)                                 //check for leap year
{
    if (!(t.year%100))
        return (char)(t.year%400);
    else
        return (char)(t.year%4);
}
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-76-58-30-00
FAX (33) 4-76-58-34-80

Atmel Programmable SLI Hotline
(408) 436-4119

Atmel Programmable SLI e-mail
fpslic@atmel.com

FAQ
Available on web site

e-mail
literature@atmel.com

Web Site
http://www.atmel.com