Introduction

The TIMEKEEPER® and serial real-time clock (RTC) devices provide an alarm which can be set either for a given time and day, or to repeat at a certain day in every month, or at a certain hour in every day, or at a certain minute of every hour, or at a certain second of every minute. With this functionality already provided in the hardware, the software to implement an alarm of any given period is greatly simplified, as described in this document.

Although specifically tailored for the M48T37V/Y device, applications can be easily adapted to use any of ST’s other TIMEKEEPER or serial RTC devices that have the alarm feature. Some modifications in the MCU memory mapping (the TIMEKEEPER or RTC address space) and in the MCU register mapping (such as the pointer to register address) would need to be made.

<table>
<thead>
<tr>
<th>TIMEKEEPER</th>
<th>M48T37V/Y, M48T201V/Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial RTC</td>
<td>M41T62, M41T63, M41T64, M41T65, M41T66, M41T80, M41T81, M41T81S, M41T82, M41T83, M41T93, M41T94, M41ST85W, M41ST87W</td>
</tr>
</tbody>
</table>
TIMEKEEPER® configuration

The TIMEKEEPER register mapping is shown in Table 2. This is divided in two parts: the clock registers and the alarm registers.

Table 2. Typical TIMEKEEPER® (M48T37V/Y) register map

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Function</th>
<th>Range (in BCD format)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7FFh</td>
<td>Year</td>
<td>Year</td>
<td>00-99</td>
</tr>
<tr>
<td>7FFEh</td>
<td>Month</td>
<td>Month</td>
<td>01-12</td>
</tr>
<tr>
<td>7FFDh</td>
<td>Date</td>
<td>Date</td>
<td>01-31</td>
</tr>
<tr>
<td>7FFCh</td>
<td>Day</td>
<td>Day</td>
<td>01-7</td>
</tr>
<tr>
<td>7FF Bh</td>
<td>Hours</td>
<td>Hour</td>
<td>00-23</td>
</tr>
<tr>
<td>7FF Ah</td>
<td>10 Minutes</td>
<td>10 Minutes</td>
<td>00-59</td>
</tr>
<tr>
<td>7FF9h</td>
<td>Seconds</td>
<td>Second</td>
<td>00-59</td>
</tr>
<tr>
<td>7FF8h</td>
<td>Calibration</td>
<td>Control</td>
<td></td>
</tr>
<tr>
<td>7FF7h</td>
<td>WDS</td>
<td>Watchdog</td>
<td></td>
</tr>
<tr>
<td>7FF6h</td>
<td>AFE</td>
<td>Interrupt</td>
<td></td>
</tr>
<tr>
<td>7FF5h</td>
<td>RPT4</td>
<td>Alarm Date</td>
<td>A Date</td>
</tr>
<tr>
<td>7FF4h</td>
<td>RPT3</td>
<td>Alarm Hour</td>
<td>A Hour</td>
</tr>
<tr>
<td>7FF3h</td>
<td>RPT2</td>
<td>Alarm Minutes</td>
<td>A Minute</td>
</tr>
<tr>
<td>7FF2h</td>
<td>RPT1</td>
<td>Alarm Seconds</td>
<td>A Second</td>
</tr>
<tr>
<td>7FF1h</td>
<td>1000 Years</td>
<td>100 Years</td>
<td>Century</td>
</tr>
<tr>
<td>7FF0h</td>
<td>WDF</td>
<td>Flags</td>
<td></td>
</tr>
</tbody>
</table>

Clock registers

The clock registers can be configured in the C language computer program as follows:

*TIMEKEEPER_CAL |= 0x80
*TIMEKEEPER_SEC= 00 //user clock setting : seconds parameter
*TIMEKEEPER_MIN= 00 //user clock setting : minutes parameter
*TIMEKEEPER_HOUR= 00 //user clock setting : hours parameter
*TIMEKEEPER_CAL&= 0x7F

The process for starting the clock and making the calibration adjustments are described in the M48T37V/Y datasheet, and in application notes AN925 and AN934.

Alarm registers

For TIMEKEEPER devices it is necessary to set the Write bit, W, at the top of the control register (at address offset 7FF8h) before proceeding to any clock modification. Modifications
to the alarm registers, though, can be made at any time, with no prior changes to the control
register being necessary.

The program listing, at the end of this document, contains statements to perform the
following functions:

1. The Stop bit (ST, bit 7 of the register at offset 7FF9h) has to be reset to start the
   TIMEKEEPER oscillator
   
   \[*TIMEKEEPER_SEC &= 0x7F; // reset bit D7 using a mask 0x7F

2. The Alarm Flag Enable (AFE) bit (bit 7 of the register at offset 7FF6h) is set, thereby
   allowing the IRQ pin (pin 40) to output the interrupt signal (active low).
   
   \[*TKPER_AL_IT |= 0x80; // set bit D7 using a mask 0x80

3. The flag register (at offset 7FF0h) must be read at the beginning of the alarm updating
   routine. If not, the AF flag will never be released, and the TIMEKEEPER will
   continuously output an interrupt to the MCU, and the system will become
   unresponsive.

Software configuration

The program is listed in Alarm update management. To understand its operation, it is
important to distinguish between the three pointer variables, pointing to physical addresses
in the hardware:

\[*TKPER_AL_HOUR, *TKPER_AL_MIN, *TKPER_AL_SEC

and the three integer variables, used as work-space by the software:

alarm_hour, alarm_minute, alarm_second

The first three variables are pointers to the physical address of the values that are stored in
the device.

The three software variables are used to hold the user’s data (they specify the period of the
alarm in hours, minutes and seconds). This is not the same information as is stored in the
TIMEKEEPER registers, as pointed to by the pointer variables, but is used in their
calculation.

The program does make use of the four Repeat bits (RPT4, RPT3, RPT2 and RPT1) that
are physically located in the TIMEKEEPER device. These should all be set, except for those
corresponding to fields that contain significant data. For instance, to set an alarm that
repeats every 3 minutes and 45 seconds, the alarm_minutes and alarm_seconds variables
would be loaded with these two values. Then appropriate values would be calculated for
loading in the “Alarm Minutes” and “Alarm Seconds” fields of the alarm registers (at
addresses 7FF3h and 7FF2h, *TKPER_AL_MIN and *TKPER_AL_SEC, respectively), and
their Repeat bits (RPT2 and RPT1, respectively) would be reset to ‘0’. Meanwhile, the
alarm_hour variable, and the “Alarm Date” and “Alarm Hour” fields of the alarm registers (at
addresses 7FF5h and 7FF4h, *TKPER_AL_DATE and *TKPER_AL_HOUR, respectively)
would be treated as “Don’t Care”, as indicated by their Repeat bits (RPT4 and RPT3,
respectively) being set. This is summarized in Table 3, with the three local integer variables,
alarm_second, alarm_minute and alarm_hour, used to represent the period.
For example, to set a period of 1 hour 49 minutes 35 seconds, the procedure is as follows:

\[
\begin{align*}
\text{RPT4} &= 1 \\
\text{RPT3} &= \text{RPT2} = \text{RPT1} = 0 \\
\text{alarm_second} &= 0x35 \\
\text{alarm_minute} &= 0x49 \\
\text{alarm_hour} &= 0x01
\end{align*}
\]

Or, to set a period of 49 minutes 35 seconds, the procedure is as follows:

\[
\begin{align*}
\text{RPT4} &= \text{RPT3} = 1 \\
\text{RPT2} &= \text{RPT1} = 0 \\
\text{alarm_second} &= 0x35 \\
\text{alarm_minute} &= 0x49 \\
\text{alarm_hour} &= \text{Don’t care}
\end{align*}
\]

Software implementation

**TIMEKEEPER® data format**

TIMEKEEPER data is held as BCD (binary coded decimal). This is handled in the C programming language using the ‘unsigned char’ data type. This can be converted within the C program to other data types, such as ‘integer’, for numeric processing. Two functions are provided in the program at the end of this document for making this conversion.

- Char_To_Int: to take a BCD parameter, and to return the equivalent integer value
- Int_To_Char: to take an integer parameter, and to return the equivalent BCD value.

The valid ranges for the alarm fields are summarized in Table 4.

<table>
<thead>
<tr>
<th>Data C language type</th>
<th>Int (integer) decimal</th>
<th>Char (character) hexadecimal</th>
<th>Char (character) BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm second</td>
<td>0-59</td>
<td>00-3B</td>
<td>0-59</td>
</tr>
</tbody>
</table>
## Alarm update management

When the alarm signal is generated by the TIMEKEEPER® device, it is communicated to the MCU. The MCU can monitor for this event either by polling, or by using interrupts. There are two variants of each method:

- **Polling**
  - Read the flag register and check the AF bit (bit 6 of the register at offset 7FF0h)
  - Output the alarm signal on the TIMEKEEPER® IRQ pin (pin 40), and read it on the MCU I/O port

- **Interrupts**
  - Give priority to processing the alarm interrupt
  - The alarm signal is used to cause a wake-up event

The last option is ideally suited when power consumption is the critical issue. For instance, when measuring, processing and storing some metering data every three minutes, the MCU can stay in its standby state for 95% of the time, and only run at full speed, with high power consumption, during the other 5% of the time.

The other interrupt option is ideally suited when service time is the critical issue. The MCU will be interrupted from whatever processing it was currently engaged in, to service the alarm event. This can be integrated into a hierarchy of prioritized interrupts.

The two polling options are equally suited when the MCU needs to run at full speed, and full power, all of the time, executing important background work, only responding to the alarm event when it has nothing else to do.

In polling method, the MCU is always running full speed and full power consumption. In this case, the application power consumption is not a key issue and/or the process to be executed due to an alarm which has no priority. The alarm check and update is served as every other application routine.

The TIMEKEEPER® IRQ pin (pin 40) is an active-low signal.

In the following program, a routine “Update_Next_Alarm” is provided to take care of the periodic update of the alarm parameters. The program has been written in ANSI C, and has been compiled and tested with an M68HC11 series MCU.

```c
/* TIMEKEEPER ; M48T37V/Y PERIODIC ALARM SOURCE CODE : */
/* Version: 1.01 */
/*****************************************************************************/
/* Copyright (c) 1999 STMicroelectronics. */
/* */
/*****************************************************************************/
/* This program is provided "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER */
/* EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTY */
/* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK */
```

### Table 4. TIMEKEEPER® data format

<table>
<thead>
<tr>
<th>Data C language type</th>
<th>Int (integer) decimal</th>
<th>Char (character) hexadecimal</th>
<th>Char (character) BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm minute</td>
<td>0-59</td>
<td>00-3B</td>
<td>0-59</td>
</tr>
<tr>
<td>Alarm hour</td>
<td>0-12</td>
<td>00-0B</td>
<td>0-12</td>
</tr>
</tbody>
</table>
/* AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE */
/* PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, */
/* REPAIR OR CORRECTION. */
*******************************************************************************/

/* This program controls the TIMEKEEPER alarm hardware so */
/* as to provide the functionality of a periodic alarm. */
*******************************************************************************/

#include <mcu_hcl1.h> // this was developed on HC11 platform
#include <m88xxfx.h> // M88 Flash+PSD register map

extern volatile unsigned char dip_sw;
*******************************************************************************/

/* TIMEKEEPER memory map */
/* Depend on your system and your TIMEKEEPER. */
/* The device is M48T37V/Y series, 32kx8 non volatile SRAM, 16 clock */
/* alarm registers in address 7FF8h to 7FFFh */
/* In this example, the TIMEKEEPER was mapped from 2000h to 9FFFh */
*******************************************************************************/

/* function Char_To_Int */
/* description : This function convert the timekeeper data*/
/* (in BCD format) to an integer. */
/* input : char byte */
/* output : integer */
/* example : octet = 0x33 (51 in integer) */
/* Char_To_Int = 33 (0x21 in hexa) */
*******************************************************************************/

int Char_To_Int(unsigned char octet)
{
  int buffer;
  buffer = (int)(octet);
  if (octet <= 0x09) return(buffer);
  if ((octet >= 0x10) & (octet <= 0x19)) return (buffer-6);
  if ((octet >= 0x20) & (octet <= 0x29)) return (buffer-12);
  if ((octet >= 0x30) & (octet <= 0x39)) return (buffer-18);
if ((octet >= 0x40) & (octet <= 0x49)) return (buffer-24);
if ((octet >= 0x50) & (octet <= 0x59)) return (buffer-30);

unsigned char Int_To_Char(int integ)
{
    char buffer;
    buffer = (unsigned char)(integ);
    if (integ <= 9) return(buffer);
    if ((integ >= 10) & (integ <= 19)) return (buffer+6);
    if ((integ >= 20) & (integ <= 29)) return (buffer+0x0C);
    if ((integ >= 30) & (integ <= 39)) return (buffer+0x12);
    if ((integ >= 40) & (integ <= 49)) return (buffer+0x18);
    if ((integ >= 50) & (integ <= 59)) return (buffer+0x1E);
}

**************************************************************************
/* function Int_To_Char */
/* description : This function convert an integer data */
/* to BCD TIMEKEEPER format {unsigned char} */
/* input : int integ */
/* output : unsigned char */
/* example : integ = 33 (0x21 in hexa) */
/* Int_To_Char = 0x33 (51 in integer) */
**************************************************************************

vised char Int_To_Char(int integ)
{
    char buffer;
    buffer = (unsigned char)(integ);
    if (integ <= 9) return(buffer);
    if ((integ >= 10) & (integ <= 19)) return (buffer+6);
    if ((integ >= 20) & (integ <= 29)) return (buffer+0x0C);
    if ((integ >= 30) & (integ <= 39)) return (buffer+0x12);
    if ((integ >= 40) & (integ <= 49)) return (buffer+0x18);
    if ((integ >= 50) & (integ <= 59)) return (buffer+0x1E);
}

**************************************************************************
/* void Update_Next_Alarm */
/* description : After alarm interupt, it will : */
/* - reset the TIMEKEEPER IT flag */
/* - read the actual time in the clock register */
/* - calculate the next alarm time */
/* - update the alarm register */
/* - to prepare for the next alarm */
/* input : alarm period {al_hour, al_minute, al_second} */
/* output : nothing */
**************************************************************************
void Update_Next_Alarm(int al_hour,int al_minute,int al_second)
{
    // time carry, going to be used for hour, minute and second
    // calculation process.
    unsigned char time_flag = 0;

    // intermediate storage for alarm data.
    int buffsec;
    int buffmin;
    int buffhour;

    // temporary storage
    unsigned char buffchar;

    // Touch the flag register to reset TIMEKEEPER AF flag (interrupt)
    buffchar = *TKPER_FLAG;

    /**************************************************************************
    /* This is to update the alarm second register. */
    /* It will test if RPT1 is set. If not then it adds "al_second" */
    /* to second alarm register. It takes care of the minute carry. */
    /**************************************************************************
    if (!(*TKPER_AL_SEC & 0x80)) // if !RPT1
    {
        // update register with carry
        buffsec = Char_To_Int(*TIMEKEEPER_SEC) + al_second;
        if (buffsec > 59) // if >59
        {
            // then restore 60sec format
            buffsec = buffsec - 60;
        }
    }
AN1216

8/11 DocID007016 Rev 2

*TKPER_AL_SEC = Int_To_Char(buffsec-60);
time_flag = 1;
}
else *TKPER_AL_SEC = Int_To_Char(buffsec); // normal case

/*****************************************************************
/* This is to update the alarm minute register. */
/* It will test if RPT2 is set. If not then it adds "al_minute" */
/* to alarm minute register. It takes care of the hour carry. */
/*****************************************************************
if (!(*TKPER_AL_MIN & 0x80)) // if !RPT2
{ // update register with carry
  buffmin = Char_To_Int(*TIMEKEEPER_MIN) + al_minute + time_flag;
  if (buffmin > 59) // if >59
    { // then restore 60 min format
      *TKPER_AL_MIN = Int_To_Char(buffmin-60);
time_flag = 1;
    }
else
  { //TKPER_AL_MIN = Int_To_Char(buffmin); // normal case
  time_flag = 0;
  }
}

/*****************************************************************
/* This is to update the alarm hour register. */
/* It will test if RPT2 is set. If not then it adds "al_hour" to */
/* alarm hour register */
/*****************************************************************
if (!(*TKPER_AL_HOUR & 80))
{
  buffhour = Char_To_Int(*TIMEKEEPER_HOUR) + al_hour + time_flag;
  if (buffhour > 23) *TKPER_AL_HOUR = Int_To_Char(buffhour-24); 
else *TKPER_AL_HOUR = Int_To_Char(buffhour);
}

main(void)
{
  int alarm_second; // relative alarm variable
  int alarm_minute;
  int alarm_hour;
  
  /*****************************************************************
  /* TIMEKEEPER alarm configuration example. */
  /*****************************************************************

  // Memory-mapped unsigned char pointers
  *TKPER_AL_HOUR &= 0x7F; // to the external hardware registers
  *TKPER_AL_MIN &= 0x7F; // to set a one-off alarm
  *TKPER_AL_SEC &= 0x7F; // for a fixed time today.

  // Local memory integer variables
alarm_hour = 1; // to hold the repetition period
alarm_minute = 49; // for an alarm (& an interrupt on pin 26)
alarm_second = 35; // every 1hr 49min 35sec (for example).

// Start the Timekeeper oscillator.
*TIMEKEEPER_SEC &= 0x7F;
// RPT4 set
*TKPER_AL_DAY |= 0x80;
// enable IRQ request on pin40 (M48T37V/Y)
*TKPER_AL_IT |= 0x80;
while (1) {
    //**************************************************************************
    /* read_the_port is a read of MCU I/O to detect an alarm interrupt */
    /* lcd_min_display is a lcd software driver used for routine debug */
    /* Those library were developed for FLASH+PSD development board. */
    //**************************************************************************
    read_the_ports();
    lcd_min_display(0,3,*TIMEKEEPER_HOUR); // display current time
    lcd_min_display(0,7,*TIMEKEEPER_MIN);
    lcd_min_display(0,13,*TIMEKEEPER_SEC);

    if (dip_sw==0x0E) // dip_sw is updated by read_the_port
        { // if detect alarm interrupt from TIMEKEEPER
            Update_Next_Alarm(alarm_hour,alarm_minute,alarm_second);

            lcd_min_display(1,3,*TKPER_AL_HOUR); // display next alarm time
            lcd_min_display(1,7,*TKPER_AL_MIN);
            lcd_min_display(1,13,*TKPER_AL_SEC);
        }
    }
}
## Revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb-2000</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>03-Sep-2013</td>
<td>2</td>
<td>Updated title of document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed references to obsolete products</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Table 1: TIMEKEEPER® and serial RTC devices with alarm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Table 2 and text throughout the document to reflect the memory map and address of the M48T37V/Y device</td>
</tr>
</tbody>
</table>
Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale. Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT AUTHORIZED FOR USE IN WEAPONS. NOR ARE ST PRODUCTS DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries. Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com