

## Creating a Real-Time Clock (RTC) with an On-chip Timer and Software

### PURPOSE

This document is a supplement to Product Change Information containing Errata for some early NXP LH75400/01/10/11 series of System-on-Chip parts. When the Errata have been corrected, the information in this Application Note will no longer apply. These parts may have a susceptibility to ground noise in the the Real-Time Clock (32.768 KHz) oscillator. They therefore may require a software work around for the Real-Time Clock (RTC), if the RTC was to be used as a time-of-day clock.

This Application Note will use one of the chip's timers to allow an application to keep track of the elapsed time in seconds or to generate an interrupt when the elapsed time matches a preset value in spite of these difficulties. These methods will work with any revision of the silicon, so the software will not need to be changed.

### INTRODUCTION

If your application does not require the features that the RTC provides, then this Application Note does not apply to you, and you may safely ground the RTC crystal input pin. If your application is always running in Active or Standby mode, see the Active or Standby mode section. If your application enters any other Low Power mode, see the Low Power Operation section.

The 32.768 kHz oscillator has a susceptibility to ground noise induced by the on-chip System Clock (HCLK) and Phase-Locked Loop (PLL). The susceptibility manifests as extra pulses on the oscillator's output. If the MCU is in Active, Standby, or Sleep mode, then the RTC Data Register 0 and RTC Data Register 1 will increment at too high a rate. How fast the clock appears to run depends on the quality of the clock crystal, the passive components, and the attention paid to the circuit board layout.

As a consequence, if the RTC is used as a time-of-day clock, the clock will run too fast.

The RTC is not recommended for use in the Sleep or Stop1 modes.

### ACTIVE OR STANDBY MODE

Unfortunately, in Active Mode and Standby Mode, the LH75400/01/10/11 PLL and HCLK are always active and therefore always producing the noise that

will corrupt the RTC. You should not use the RTC if your application only uses Active or Standby Mode. This part of the work around assumes that your application has these properties:

- The application requires a way to track the elapsed time in seconds (e.g., the time-of-day), and/or it needs to generate an interrupt when the elapsed time matches a pre-determined elapsed time.
- The application is running an operating system that supports software timers, or the application is not using all three timers that are clocked using HCLK (Timer 0, Timer 1, and Timer 2), or the application is using Timer 0, Timer 1, or Timer 2 to generate a periodic waveform or periodic interrupt.
- The application never stops the timer that is generating the operating system clock, or in a non-operating system application, the application never stops the timer that is generating a periodic waveform or periodic interrupt.

### Software Algorithm

Declare a 32-bit global variable that your application will use to keep track of the elapsed time in seconds:

```
rtc_elapsed_time; // UNS_32: 32-bit unsigned type
```

Initialize the 32-bit global variable with the same value you would have used to initialize the RTC load register.

If your application requires the same feature provided by the RTC match register, (i.e., Match Register 0 and Match Register 1), declare a second 32-bit global variable to store the match value:

```
UNS_32 rtc_match;
```

If your application uses an operating system with a timer facility, use that facility to increment the 32-bit global variable every second. For example, set up a high priority task that wakes every second, increments the variable, and then suspends.

For lower overhead if your application's operating system provides hooks to the timer interrupt, or if your application doesn't have an operating system, program your application's periodic timer interrupt to increment your elapsed time global variable after the appropriate number of timer interrupts have occurred.

If you require the interval match facility, use the software interrupt feature of the Vectored Interrupt Controller (VIC) to assert the RTC interrupt.

This example generates a timer interrupt rate of 100 interrupts per second: (These C examples assume that the interrupt handler functions are called by a wrapper function

that saves context on entry and restores context on exit).

Note that the HCLK is still running in Standby mode, so even if your application enters Standby mode, the timers will still run and the timer interrupt will still move the MCU from Standby to Active mode.

### TIMER INTERRUPT EXAMPLE

```
/* Modifications to the periodic timer interrupt handler */
#define TIMER_INTS_PER_SECOND 100
#define VIC_SOFT_INT_PTR ((volatile UNS_32 *)0xFFFFF018)
#define VIC_SOFT_INT_CLR_PTR ((volatile UNS_32 *)0xFFFFF01C)
#define RTC_ALARM 15
void timer_int_handler(void)
{
    static UNS_32 rtc_ints_remaining = TIMER_INTS_PER_SECOND;
    /* other timer interrupt declarations go here */

    if (0 == --rtc_ints_remaining)
    {
        rtc_elapsed_time++;
        rtc_ints_remaining = TIMER_INTS_PER_SECOND;
        if (rtc_elapsed_time == rtc_match)
        {
            *VIC_SOFT_INT_PTR = 1 << RTC_ALARM;
        }
    }

    /* do what the timer interrupt normally does */
}

/* RTC Alarm Software Interrupt */
void rtc_soft_alarm(void)
{
    /* clear the alarm interrupt */
    *VIC_SOFT_CLR_PTR = 1 << RTC_ALARM;

    /* do the processing for the alarm */
}
```

## LOW POWER OPERATION

The application that requires support from the RTC more than any other is the battery-powered application that needs to keep track of the time of day and date even when every other chip function is off. The LH75400/01/10/11 provides three very low power modes: Standby, Stop1 and Stop2. Due to the RTC oscillator errata, the only Low Power mode available to use this function is Stop2. Fortunately for battery-powered applications, Stop2 is the lowest Power mode of all these Soc's operating modes.

In Active or Standby mode, use the algorithms to track time described previously in 'Active or Standby mode'. Use this algorithm with the RTC to count elapsed seconds in Low Power mode:

1. Copy the global variable that counts elapsed time into the RTC load register (i.e., copy the least significant 16 bits into Load Register 0, and then copy the most significant 16 bits into Load Register 1).
2. If your application needs to use the RTC Match Interrupt to exit, copy the Global Elapsed Seconds Match variable into the RTC match register (i.e., copy the least significant 16 bits into Match Register 0, and then copy the most significant 16 bits into Match Register 1).
3. Poll the RTC Data Register until it contains a value that is 1 more than the value you loaded into the RTC Match Register. This step ensures that the value you wrote to the RTC Load Register has been transferred from the HCLK domain into the RTC clock domain.
4. Write 0b100 to the PWR DWN SEL bit field of the Reset, Clock and Power Controller (RCPC) Control Register to enter Stop2 mode. Make sure the lock bit of the RCPC Control Register is set before trying to write to the RCPC or else writes to the RCPC will be ignored.

### RTC STOP2 MODE ENTRY EXAMPLE

```
#define RTC_DATA0_PTR ((volatile UNS_32 *)0xFFFE0000)
#define RTC_DATA1_PTR ((volatile UNS_32 *)0xFFFE0004)
#define RTC_LOAD0_PTR ((volatile UNS_32 *)0xFFFE0014)
#define RTC_LOAD1_PTR ((volatile UNS_32 *)0xFFFE0018)
#define RTC_MATCH0_PTR ((volatile UNS_32 *)0xFFFE0008)
#define RTC_MATCH1_PTR ((volatile UNS_32 *)0xFFFE000C)
#define RCPC_CTRL_PTR ((volatile UNS_32 *)0xFFFE2000)
#define RCPC_PWR_DWN_SEL_BIT 2
#define RCPC_STOP2_VAL 4
#define RCPC_LOCK_BIT 9
#define HALFWORD_MASK 0xFFFF
#define HALFWORD_SIZE 16
void enter_stop2_mode(void)
{
    /* make sure the RCPC is unlocked */
    *RCPC_CTRL_PTR |= (1 << RCPC_LOCK_BIT);

    /* prepare the RTC registers */
    *RTC_LOAD0_PTR = rtc_elapsed_time & HALFWORD_MASK;
    *RTC_LOAD1_PTR = rtc_elapsed_time >> HALFWORD_SIZE;
    *RTC_MATCH0_PTR = rtc_match & HALFWORD_MASK;
    *RTC_MATCH1_PTR = rtc_match >> HALFWORD_SIZE;

    /* make sure the load value has transferred to the RTC Data Register */
    while (*RTC_DATA0_PTR < rtc_elapsed_timer + 1)
        ; /* wait */

    /* enter Stop2 mode */
    *RCPC_CTRL_PTR |= RCPC_STOP2_VAL << RCPC_PWR_DWN_SEL_BIT;
}
```

To exit Stop2 mode, use either an external interrupt (one of the INT0-INT6 pins) or the RTC\_ALARM interrupt. The interrupt handler that exits Stop2 mode should do the following:

- Clear the interrupt that caused the exit from Stop2 mode.
- Read the RTC Data Register until the value read from the RTC Data Register is greater than the value that was written prior to entering Stop2 mode.
- Write this value into the global variable that counts Elapsed Time.

- Perform any other processing the interrupt service routine needs to do.
- Exit the interrupt. At this point, keep track of the elapsed time using the procedure described in 'Active or Standby mode'.

#### RTC STOP2 MODE EXIT EXAMPLE

This example exits Stop2 mode on RTC\_ALARM interrupt: (This assumes the interrupt handler is called by a wrapper that saves the registers on entry and restores them on exit.)

```
#define RTC_EOI_PTR ((volatile UNS_32 *)0xFFFFE0010)
void rtc_alarm_handler(void)
{
    UNS_32 new_time;

    /* clear the RTC_ALARM interrupt */
    *RTC_EOI_PTR = 1;

    /* get the time from the RTC */
    do
    {
        new_time = *RTC_DATA0_PTR;
        new_time |= *RTC_DATA1_PTR << HALFWORD_SIZE;
    } while (new_time < rtc_elapsed_time + 1);
    rtc_elapsed_time = new_time;

    /* do processing required by the alarm */
}
```

## LIMITATIONS

For most applications that require a real-time clock, the software methods described here will allow acceptable performance.

Applications that are using all three on-chip timers for non-periodic timing functions (e.g., are only used to time brief intervals or are used with an aperiodic external clock) cannot use this software method.

Your application cannot use the RTC ALARM function in Active mode or Standby mode.

Your application cannot use the RTC with Sleep or Stop1 mode.

Your application will take as long as 1 second to enter and exit Stop2 mode due to RTC synchronization.

Your application could gain as much as 1 second of error in the elapsed time every time it enters or exits Stop2 mode.

# ANNEX A: Disclaimers (11)

## 1. t001dis100.fm: General (DS, AN, UM, errata)

**General** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

## 2. t001dis101.fm: Right to make changes (DS, AN, UM, errata)

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

## 3. t001dis102.fm: Suitability for use (DS, AN, UM, errata)

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

## 4. t001dis103.fm: Applications (DS, AN, UM, errata)

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

## 5. t001dis104.fm: Limiting values (DS)

**Limiting values** — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) may cause permanent damage to the device. Limiting values are stress ratings only and operation of the device at these or any other conditions above those given in the Characteristics sections of this document is not implied. Exposure to limiting values for extended periods may affect device reliability.

## 6. t001dis105.fm: Terms and conditions of sale (DS)

**Terms and conditions of sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <http://www.nxp.com/profile/terms>, including those pertaining to warranty, intellectual property rights infringement and limitation of liability, unless explicitly otherwise agreed to in writing by NXP Semiconductors. In case of any inconsistency or conflict between information in this document and such terms and conditions, the latter will prevail.

#### 7. t001dis106.fm: No offer to sell or license (DS)

**No offer to sell or license** — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

#### 8. t001dis107.fm: Hazardous voltage (DS, AN, UM, errata; if applicable)

**Hazardous voltage** — Although basic supply voltages of the product may be much lower, circuit voltages up to 60 V may appear when operating this product, depending on settings and application. Customers incorporating or otherwise using these products in applications where such high voltages may appear during operation, assembly, test etc. of such application, do so at their own risk. Customers agree to fully indemnify NXP Semiconductors for any damages resulting from or in connection with such high voltages. Furthermore, customers are drawn to safety standards (IEC 950, EN 60 950, CENELEC, ISO, etc.) and other (legal) requirements applying to such high voltages.

#### 9. t001dis108.2.fm: Bare die (DS; if applicable)

**Bare die (if applicable)** — Products indicated as Bare Die are subject to separate specifications and are not tested in accordance with standard testing procedures. Product warranties and guarantees as stated in this document are not applicable to Bare Die Products unless such warranties and guarantees are explicitly stated in a valid separate agreement entered into by NXP Semiconductors and customer.

#### 10. t001dis109.fm: AEC unqualified products (DS, AN, UM, errata; if applicable)

**AEC unqualified products** — This product has not been qualified to the appropriate Automotive Electronics Council (AEC) standard Q100 or Q101 and should not be used in automotive critical applications, including but not limited to applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

#### 11. t001dis110.fm: Suitability for use in automotive applications only (DS, AN, UM, errata; if applicable)

**Suitability for use in automotive applications only** — This NXP Semiconductors product has been developed for use in automotive applications only. The product is not designed, authorized or warranted to be suitable for any other use, including medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.