Embedding it better...

µTasker Document

µTasker – Time-Keeping

Table of Contents

1. Introduction

Keeping track of time is of importance in many projects. This can mean simply knowing how much time has passed or it can also mean knowing the relationship to the absolute time and date at a certain location. Furthermore the required accuracy can vary greatly, from sub-µs to just rough ideas about the time. Finally, the time may need to be configured or requested regularly from external time sources, it may also be necessary to be able to keep monitoring accurate time during long periods or absence of power; either processor power or all power sources.

This document describes the time-keeping support included in the µTasker project, which includes the following topics:

- Second counting and transforming to Gregorian time (and relationship to UTC)
- Internal Real-Time-Clocks
- External Real-Time-Clocks
- Time sources via Internet (Time Server and Simple Network Time Protocol [SNTP])

2. Seconds Counter

Probably the most useful and simplest method of keeping track of time is by using a counter which is incremented once a second. Knowing the number of seconds that have passed since a defined point in time allows the present date and time to also be calculated to an accuracy of a second. The simplest RTCs (Real Time Clocks) are nothing more than a counter that is incremented at a one second rate; in comparison, a more complex RTC may contain various counters to represent seconds, minutes, hours, day, day of week, month and year, as well as other information that may be useful in relation to time and date – such as leap-years and daylight saving. RTCs are discussed in more detail in subsequent sections.

Thursday, 01.01.1970 is often used as the reference for seconds counting. This is called the coordinated universal time (UTC) and the number of second elapsed since then (not counting leap seconds) is often referred to as Unix time, Epoch seconds or POSIX time.

Usually 32 bits are used for the seconds counter which allows operation up to around 139 years (about 2109) and for each date and time in that period an equivalent seconds count value exists and for each seconds count value a date and time can be represented.

Assuming a system in which the seconds counter value is invalid each time that the system is started it is necessary to first request the time and date or else have a user enter this. The entered value can be saved to the counter in its equivalent converted form and then the system needs to simply increment the variable once a second using a timer source. When the user requests the time, or in order to display it, the seconds counter value then needs to be converted back to its date and time representation. This gives a small set of minimal methods for a second counter to be able to operate satisfactory:

- Accurate seconds timer to increment the seconds variable
- Method to input a time and date and save this to the seconds variable
- Method to output the present seconds value as a time and date

Also when there is a RTC available it may be convenient to support the conversion methods, whereby the RTC is usually responsible for supplying an accurate seconds interrupt. Since RTCs are often battery backed-up the time continues to be updated when the system is otherwise powered down and so there is no need to input or request the time each time the system restarts.

The µTasker software based RTC implementation allows the time/date to be retained across software resets – to an accuracy of about 1s – by maintaining the seconds count in an area of SRAM that is not initialised. Thsi means that after a software reset the value is not reset and the seconds counter can continue to be used as long as there was no power loss and so corruption of the value (which is further checked with a validity pattern also maintained in the same area of SRAM).

3. Internal Real-Time-Clock

Various processors that are supported by the µTasker project contain an internal RTC (real time clock). They are typically characterised by the fact that they generate a one second clock (some from their own 32kHz crystal oscillator but some from the main processor clock) and some have a set of registers to help represent time and date (in the simplest case this is a second counter as described in the previous section but may involve various registers). They often have their own supply pin so that the RTC can continue running from a battery when the main processor power has been removed, as long as the clock continues to operate as well. Low power RTCs can continue to operate from their battery back-up supply at low voltages (eg. 2V) and with low power consumption (often less that 1uA for the RTC and its clock).

Not all processors containing a RTC will have a backup power capability for it, however processors which support very low power modes can still be used as real time clock systems when the main power can be reduced to a level close to that achieved by stand-alone RTCs. In such a state the RTC is still operating although the processor is itself in a standby state; a possible advantage of such a state is that it can be executed by an alarm in the RTC and so recover to running mode at a pre-determined time.

Since the features offered by each processor type can vary greatly each supported RTC is described in more detail to illustrate its strengths and/or weaknesses. The RTC user's interface has been designed for a high degree of compatibility, whereby the interface may add additional functionality to the RTC features that are actually available in the peripheral hardware.

3.1. M522XX

M521XX, M5221X and M5225X parts have battery backup for RTC and SRAM. They have an on-chip 32kHz oscillator with dedicated pins on the M5225X, or share these with UART0 on the M521XX and M5221X. As long as the RTC is continuously supplied with the correct voltage the RTC will maintain time when the processor is otherwise powered down.

Other parts often have a non-battery back RTC where its 1 Hz clock is derived from the processor's clock by dividing it down by the correct ratio. These types do not hold time and date information when the processor is powered down.

3.2. Kinetis K Family RTC

Kinetis K family parts all contain a RTC which is a simple 32 bit counter clocked from a defined source, including internal RC oscillators or optionally from dedicated 32kHz oscillators in some cases. A 1Hz interrupt from the counter can be generated by using its alarm match capability or using a dedicated 1Hz interrupt offered in some of the parts (newer versions tend to have a 1Hz interrupt as well as the alarm match interrupt).

The RTC has its own power domain and can therefore continue operating when battery backup is used and the processor itself is disconnected from the supply.

3.3. Kinetis KL Family RTC

The Kinetis KL family parts are characterised as using the same RTC as the K family parts but without the battery backup. Furthermore the RTC is not generally clocked from the 32kHz oscillator when the processor is in very low power modes, meaning that applications may require an external low power oscillator to be used instead in order to achieve best performance.

Some additional notes about the RTC in the KL parts can be found at http://www.utasker.com/kinetis/KL_RTC.html.

3.4. Kinetis KE Family RTC

The Kinetis KE family parts are characterised as having a Real-Time-Counter (as opposed to a Real-Time-Clock). This allows an accurate 1s interrupt to be generated when the processor is running (or in some low power modes where the interrupt can temporarily wake the processor to be handled). The RTC doesn't contain and registers to hold the time and is reset when the processor is reset.

By using the software based RTC mode, based on 1s interrupt, the KE can be used successfully in systems that remain powered and time/date retained across software resets.

4. External Real-Time-Clock

A popular and very low power external RTC is the Dallas (Maxim) DS1307 which has its own 32kHz oscillator and connects to a processor via an I^2C bus. It maintains time when the processor is powered down as long as it has its own battery backup, whereby less that 500nA current consumption is typically required.

The RTC can output a 1Hz square wave which can be used by a processor as 1 second interrupt.

5. RTC Software Interface

The µTasker project framework supports time keeping using its Time Keeper interface in

`time_keeper.c`

This contains a collection of interfaces for operations with various RTCs and time protocols, as well as routines that allow setting and displaying time, date and alarms or conversions between UTC and local time.

5.1 Configuring/Starting the RTC

The first time that an application is started on a board that hasn't been powered before the RTC will generally not be operating and will also have no valid time/date. Each time an application starts it should therefore call the RTC initialisation function:

```
extern void fnStartRTC(void (*seconds_callback)(void));
```

This function will cause the state of the RTC to be checked and a basic configuration will be performed if it has not yet been started. A parameter value of zero means that the application doesn't require a seconds call back to be executed:

```
fnStartRTC(0);
```

In case the application prefers to have a seconds interrupt callback it can pass a pointer to the call back function that should be used.

Some RTCs require a stabilising period for their oscillator when enabled for the first time. This function will always returns immediately but the time keeper task may allow the stabilisation period to pass before the final settings are actually performed.

5.2 Reading and Displaying the Time/Date

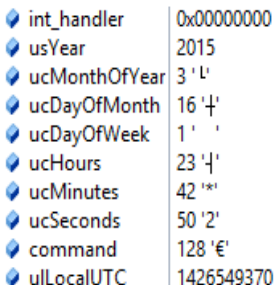
The time/date is read using the function

```
extern void fnGetRTC(RTC_SETUP *ptrSetup);
```

For example:

```
RTC_SETUP time_date;
fnGetRTC(&time_date);
```

The `RTC_SETUP` struct is filled with details of the present time and date relative to the user (local time). Notice that there is a seconds count value in the field `ulLocalUTC` as well as field for representation in minutes, hours, day of week etc. The name `ulLocalUTC` avoids confusion with explicit UTC since it is only equal to UTC when the local time is UTC, otherwise it includes an offset for the local time zone and daylight saving time.



The interface also supplies methods to convert the time and data for display purposes using

```
extern unsigned char fnSetShowTime(int iSetDisplay, CHAR *ptrInput);
```

Example:

```
CHAR cTimeDateBuf[20];
fnSetShowTime(DISPLAY_RTC_TIME_DATE, cTimeDateBuf);
fnDebugMsg(cTimeDateBuf);
fnDebugMsg("\r\n");
```

which sends the string to the debug output as "`16.03.2015 23:42:50`"

5.3 Setting the Time/Date

The values in a RTC_SETUP object can be modified as desired and then committed to the RTC by using the function

```
extern int fnConfigureRTC(void *ptrSettings);
```

For example:

```
RTC_SETUP rtc_setup;
rtc_setup.command = RTC_TIME_SETTING;
rtc_setup.usYear = 2015;
rtc_setup.ucMonthOfYear = 2;
rtc_setup.ucDayOfMonth = 14;
rtc_setup.ucHours = 12;
rtc_setup.ucMinutes = 30;
rtc_setup.ucSeconds = 0;
fnConfigureRTC(&rtc_setup); // write the date/time to the RTC
```

This function will commit the new date/time to the RTC that the HW uses.

The field ucDayOfWeek does not need to be prepared since it will be calculated by the method and can be subsequently read back if of interest, as is the same for the local UTC value.

In case UTC (referenced to the local time) is available instead of the date and time values as above this can be used to set the RTC's values by calling the function as follows:

```
RTC_SETUP rtc_setup;
rtc_setup.command = (RTC_TIME_SETTING | RTC_SET_UTC);
rtc_setup.ulLocalUTC = 1426491254;
fnConfigureRTC(&rtc_setup); // write the date/time to the RTC (based on UTC
value)
```

In this case the passed local UTC value is used for the configuration and the corresponding date/time can be read back if of interest.

In case of the need to convert between UTC and Gregorian date/time without actually setting the RTC the same interface can be used but with the flags RTC_CONVERT_TO_UTC or RTC_CONVERT_FROM_UTC instead of RTC_TIME_SETTING. Examples of this are given in following section.

Since date and time will often be entered as ASCII strings the time keeper interface allows for setting these to the RTC as follows based on:

```
extern unsigned char fnSetShowTime(int iSetDisplay, CHAR *ptrInput);
```

Example:

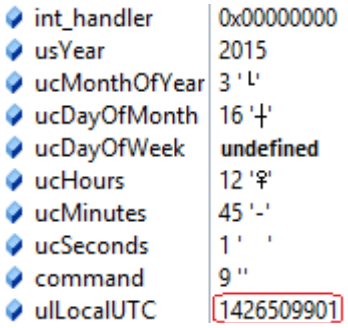
```
    fnSetShowTime(SET_RTC_TIME, "20:15:00"); // set the time
    fnSetShowTime(SET_RTC_DATE, "3.2.2015"); // set the date
```

Note that date entries can also use the forms 3:02:2015 or 3.2.15 etc. where dates previous to the year 2000 are assumed never to be required.

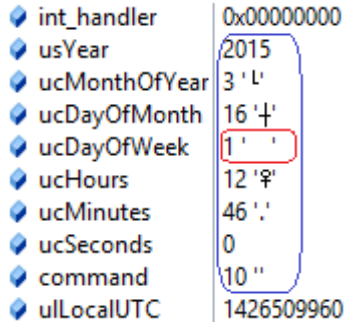
5.4 Converting between Gregorian date/time and UTC

The interface described for setting the date and time in the previous section can also be used for general conversions without affecting the RTC value. The following two uses show how UTC can be converted to Gregorian date/time and then back to UTC.

```
RTC_SETUP rtc_setup;
rtc_setup.command = RTC_CONVERT_TO_UTC;
rtc_setup.usYear = 2015;
rtc_setup.ucMonthOfYear = 3;
rtc_setup.ucDayOfMonth = 16;
rtc_setup.ucHours = 12;
rtc_setup.ucMinutes = 45;
rtc_setup.ucSeconds = 1;
fnConfigureRTC(&rtc_setup); // date/time to UTC
```




```
 RTC_SETUP rtc_setup;
rtc_setup.command = RTC_CONVERT_FROM_UTC;
rtc_setup.ullocalUTC = (1426509901 + 59);
fnConfigureRTC(&rtc_setup); // UTC to date/time
```



6. Simple Network Time Protocol

Devices connected to a network (usually via Ethernet) can request the present time using SNTP as supported in the µTasker project and described in the SNTP document http://www.utasker.com/docs/uTasker/uTaskerSNTP.pdf

Acting as an SNTP client, the device requests the time from an SNTP server either in the local network or in the Internet. The received time is then used to set or synchronise the local RTC.

It is to be noted that the RTC operates directly as the user's time. An example would be 12:21:32 on a Monday afternoon since this is the local time as interpreted by the user. The local time may be using daylight saving if it happens to be in the middle of winter and would be typically 6 hours behind the time in London if the user happened to be located in Dallas, Texas. SNTP will return always UTC and not the local time and so both the time zone and daylight saving has to be considered when setting the RTC's value! UTC is referenced to the time in London without daylight saving and the conversion is to simply add the time zone compensation (-21'600 seconds for Dallas when 6 hours behind) and a further -3600 seconds in case daylight saving is presently active.

SNTP references time to 1.1.1900 and so is not directly a UTC representation (referenced to 1.1.1970) so the SNTP time value (seconds) first needs to be adjusted to UTC seconds by subtracting 2'208'988'800 seconds (the number of seconds from 1.1.1900 to 1.1.1970).

Note that there are 34 different time zones in the world. London is at UTC+0 and most time zones are at differences divided by one hour, ranging from -12 hours to +13 hours. There are however a few exceptions:

- UTC-4:30 Caracas
- UTC-3:30 Newfoundland
- UTC+3:30 Tehran
- UTC+4:30 Kabul
- UTC+5:30 New Delhi
- UTC+5:45 Kathmandu
- UTC+6:30 Rangoon
- UTC+9:30 Darwin

7. Conclusion

This document has discussed various RTC techniques possible and available in supported processors. The µTasker Time Keeping interface has been introduced and its application interface, with basic compatibility irrespective of the underlying hardware's capabilities, has been detailed.

Modifications:

V0.00 16.03.2015:
- Partial draft version for the V1.4.8 project

V0.01 15.04.2015:
- Updated to explain the Kinetis KE real-time-counter