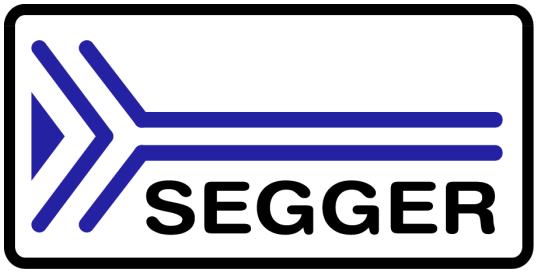
# Application Note

# Using embOS tickless support with STM32

Document: AN01002

Revision: 0

Date: June 26, 2014




A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

## Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH & Co. KG (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

## Copyright notice



## Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

## Manual versions

This manual describes the current software version. If any error occurs, inform us and we will try to assist you as soon as possible.
Contact us for further information on topics or routines not yet specified.

Print date: June 26, 2014

| Revision | Date | By | Description |
|---|---|---|---|
| 0 | 140626 | TS | First version. |

# Abstract

This application note describes the usage of the embOS tickless support with a STM32F103 CPU.

# Introduction

Stopping the system tick interrupt allows the microcontroller to remain in a deep power saving state until an interrupt occurs. This helps to save power for e.g. battery powered devices.

The embOS tickless support stops the periodic system tick interrupt during idle periods. Idle periods are periods of time when there are no tasks or software timer ready for execution.

# Implementation of OS_Idle()

We use instead of the generic Cortex-M systick timer the hardware timer TIM2. We use the timer in up-counting mode.

In order to use the tickless support the OS_Idle() function has to be modified. The default OS_Idle() function is just an endless loop which starts the low power mode:

```
void OS_Idle(void) {
  while (1) {
    __asm(" wfi");
  }
}
```

The tickless OS_Idle() function calculates the amount of time which should be spent in low power mode and reprograms the timer accordingly:

```
void OS_Idle(void) {
  OS_TIME IdleTicks;
  OS_DI();
  IdleTicks = OS_GetNumIdleTicks();
  if (IdleTicks > 1) {
    if ((OS_U32)IdleTicks > TIM2_MAX_TICKS) {
      IdleTicks = TIM2_MAX_TICKS;
    }
    OS_StartTicklessMode(IdleTicks, &_EndTicklessMode);
    TIM2_ARR = (OS_TIMER_RELOAD * IdleTicks) - TIM2_CNT; // Set compare reg.
  }
  OS_EI();
  while (1) {
    __asm(" wfi");
  }
}
```

**The following description analyzes the OS_Idle() function step by step:**

```
void OS_Idle(void) {
  OS_TIME IdleTicks;
  OS_DI();
```

Interrupts are disabled to avoid a timer interrupt.

```
IdleTicks = OS_GetNumIdleTicks();
  if (IdleTicks > 1) {
```

The OS_Idle() function reads the idle ticks with OS_GetNumIdleTicks(). The tickless mode is only enabled when there is more than one idle tick. If there are zero or one idle ticks the scheduler is executed at the next system tick hence it makes no sense to enable the tickless mode.

```
if ((OS_U32)IdleTicks > TIM2_MAX_TICKS) {
  IdleTicks = TIM2_MAX_TICKS;
}
```

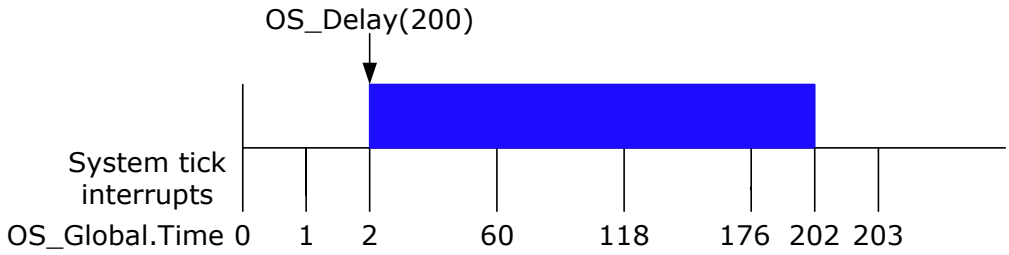
The STM32F103 hardware timer TIM2 is a 16bit timer. We set the clock prescaler register TIM2_PSC to 63 which devides the peripheral clock by 64. The CPU runs at OS_FSYS = 72 MHz.

Hence the maximum count time is:

T = (0xFFFF x (TIM2_PSC + 1)) / OS_FSYS  = 0.058253 sec ~ 58 msec

This setup uses 1 msec per system tick. Thus we can stay at most for 58 system ticks in low power mode.

If idle ticks are greater than this maximum value we just set idle ticks to 58. For example OS_GetNumIdleTicks() returns 200. The tickless mode will be enabled for 58 system ticks. The next time OS_Idle() is executed OS_GetNumIdleTicks() returns 142. The tickless mode will again be enabled for 58 system ticks for another two times. The last time OS_GetNumIdleTicks() returns 26.




Instead of having 200 interrupts (each one for each system tick) we need only 4 timer interrupts. The system can stay in low power mode for a much longer time which saves power.

```
if (IdleTicks > 1) {
  ...
  OS_StartTicklessMode(IdleTicks, &_EndTicklessMode);
  TIM2_ARR = (OS_TIMER_RELOAD * IdleTicks) - TIM2_CNT; // Set compare register
}
```

OS_StartTicklessMode() sets the idle ticks and the callback function. The idle ticks information is later used in the callback function. The callback function is described below. We adjust the timer compare register to the new calculated value. Since the timer is already running we have to subtract the current timer value.

```
_OS_EI();
while (1) {
  __asm(" wfi");
}
```

Interrupts are reenabled and the CPU enters in the endless while loop the low power mode.

# Implementation of the callback function()

The callback function calculates how long we actually stayed in low power mode and corrects the system time accordingly. The hardware timer will be reset to the default system tick time of 1 msec.

```
static void _EndTicklessMode(void) {
  OS_U16 NumTicks;
  OS_U16 Cnt;
  OS_U16 IReq;

  if (OS_Global.TicklessExpired) {
    // The timer interrupt was executed => we completed the sleep time
    OS_AdjustTime(OS_Global.TicklessFactor);
  } else {
    Cnt  = TIM2_CNT;
    IReq = TIM2_SR & (1uL << 0u);
    if (IReq) {
      OS_AdjustTime(OS_Global.TicklessFactor);
    } else {
      //
      // We assume OS_TIMER_RELOAD Counts per tick and hardware timer
      // which counts up.
      //
      NumTicks = Cnt / OS_TIMER_RELOAD;
      Cnt     -= (NumTicks * OS_TIMER_RELOAD);
      TIM2_CNT = Cnt;
      OS_AdjustTime(NumTicks);
    }
  }
  TIM2_ARR = OS_TIMER_RELOAD; // Set default value for 1 tick
}
```

**The following description analyzes the callback function step by step:**

```
static void _EndTicklessMode(void) {
  ...
  if (OS_Global.TicklessExpired) {
    OS_AdjustTime(OS_Global.TicklessFactor);
```

If the hardware timer has expired and the system tick interrupt was executed the flag OS_Global.TicklessExpired is set to 1. This can be used to determine if the system stayed in low power mode for the complete idle time. If this flag is set we can use the value in OS_Global.TicklessFactor to adjust the system time.

```
  } else {
    Cnt  = TIM2_CNT;
    IReq = TIM2_SR & (1uL << 0u);
    if (IReq) {
      OS_AdjustTime(OS_Global.TicklessFactor);
```

The same is true when the system tick interrupt was not executed but the interrupt pending flag in the timer status register is set. We adjust the system time with OS_Global.TicklessFactor.

```
} else {
      NumTicks = Cnt / OS_TIMER_RELOAD;
      Cnt     -= (NumTicks * OS_TIMER_RELOAD);
      TIM2_CNT = Cnt;
      OS_AdjustTime(NumTicks);
}
```

This last else branch is executed when the scheduler was triggered by another interrupt than the timer interrupt. In this case we calculate how many system ticks did the system actually spent in low power mode. We adjust the timer count register and the system time accordingly.

```
TIM2_ARR = OS_TIMER_RELOAD; // Set default value for 1 tick
```

The timer compare register is reset to default value for one system tick which is equal to 1 msec.