This application note describes managing the Talaria TWO watchdog timer using the functions provided by the watchdog driver.

# Watchdog Timer

The watchdog driver provides functions for managing the Talaria TWO watchdog timer.

It is a down-counting timer that generates a system reset (or an interrupt) when the counter reaches zero. The timer is started with an application specific timeout via the watchdog\_init() and watchdog\_start() functions. The client application uses the watchdog\_kick() function to periodically reload the counter to its initial value to avoid it counting to zero.

Depending on what the application wants to supervise using this watchdog, the watchdog\_kick() may be performed from contexts. If the goal is to make sure that a certain real-time response time is required on a particular thread priority level, a dedicated thread on this priority level should kick the watchdog. Failure to meet the real-time response will then lead to a system reset. Another use case is to supervise on interrupt level. In this case, the watchdog should be kicked from an interrupt service handler on a timer interrupt that is scheduled at an interval slightly lower than the watchdog timeout.

# Sample Code Walkthrough

The sample application provides functions for managing the Talaria TWO watchdog timer.

**wdreset.c**

The callout\_init()function initiates the callout object. wakeup callout is used to schedule a wakeup from suspend state. xSemaphoreCreateCounting(1, 0) initiates the semaphore with a default value of 0. watchdog\_init initializes the watchdog with the specified timeout in microseconds and the watchdog expire\_cb is passed as NULL, resulting in a system reset at timer expiration. The watchdog\_start() function starts the watchdog counter.

|  |
| --- |
| callout\_init(&wakeup, wakeup\_callback);  wakeup\_event = xSemaphoreCreateCounting(1, 0);  watchdog\_init(WATCHDOG\_TIMEOUT, NULL);  watchdog\_start(); |

callout\_init\_soft()initializes callout object with a soft deadline. kicker callout is used to kick the watchdog to reload the counter to its initial value to avoid the counter value reaching zero. A soft callout timeout accepts the timeout to be invoked later than requested if this could lead to savings in power consumption. The callout\_schedule()schedules the callback function to be invoked after the specified number of microseconds. Application needs to kick the watchdog before it expires, hence the timeout is given as WATCHDOG\_TIMEOUT \* 9 / 10.

|  |
| --- |
| callout\_init\_soft(&kicker, kick\_the\_dog);  callout\_schedule(&kicker, WATCHDOG\_TIMEOUT \* 9 / 10); |

Talaria TWO is put to suspend mode to demonstrate that the watchdog works fine both in suspend mode and awake mode. After enabling the suspend mode, wakeup callout is scheduled after two seconds. Then the thread suspends on a wakeup\_event semaphore. Since there is no other activity scheduled at this point, the system enters suspend mode.

|  |
| --- |
| os\_suspend\_enable();  callout\_schedule(&wakeup, SYSTIME\_SEC(2));  os\_printf("Sleeping\n");  if (xSemaphoreTake(wakeup\_event, portMAX\_DELAY) == pdFAIL) {  os\_printf("Unable to wait on semaphore...!!\n");  return -1;  }  os\_printf("Awake\n");  os\_suspend\_disable();  vTaskDelay(4000); |

This callback function of wakeup callout wakes the system up from suspend state and posts the semaphore for which the main thread was waiting before going to suspend mode.

**Note**: wakeup\_callback is executed from ISR context, hence the SemaphoreGiveFromISR() is used.

|  |
| --- |
| static void  wakeup\_callback(struct callout \*c)  { xSemaphoreGiveFromISR(wakeup\_event, NULL); } |

This callback function of kicker callout kicks the watchdog.

|  |
| --- |
| static void  kick\_the\_dog(struct callout \*c)  {  static uint32\_t num\_kicks;  os\_printf("Kick\n");  watchdog\_kick();  if (++num\_kicks < 30)  callout\_schedule(c, WATCHDOG\_TIMEOUT \* 9 / 10);  else  os\_printf("Last kick\n");  } |

Here, kick\_the\_dog()function calls watchdog\_kick()before the watchdog timer expires.

Every time watchdog\_kick() is called, it resets the watchdog timer which is set using watchdog\_init(). After the 30th call of watchdog\_kick(), this callback is not scheduled and thus the watchdog timer expires, resulting in system reset.

System reset occurs because the watchdog\_init() was called with expire\_cb passed as NULL.

|  |
| --- |
| watchdog\_init(WATCHDOG\_TIMEOUT, NULL); |

## Running the Application

Program wdreset.elf (*freertos\_sdk\_x.y\examples\watchdog\_timer\bin*) using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down.
   2. ELF Input: Load the wdreset.elf by clicking on Select ELF File.
   3. Programming: Prog RAM or Prog Flash as per requirement.

**Note**: Post 30th call of the watchdog\_kick()function, the watchdog timer expires

* 1. Prog RAM: After reset, wdreset.elf halts
  2. Prog Flash: After reset, wdreset.elf runs in a loop

## Expected Output

|  |
| --- |
| Build $Id: git-df9b9ef $  Flash detected. flash.hw.uuid: 39483937-3207-0051-002a-ffffffffffff  $App:git-6818774  SDK Ver: FREERTOS\_SDK\_1.0  Watchdog Reset Demo App  Starting watchdog  Sleeping  Kick  Kick  Awake  Kick  Kick  Kick  Kick  Sleeping  Kick  Awake  Kick  Kick  Kick  Kick  Sleeping  Kick  Awake  Kick  Kick  Kick  Kick  Sleeping  Kick  Awake  Kick  Kick  Kick  Kick  Sleeping  Kick  Awake  Kick  Kick  Kick  Kick  Sleeping  Kick  Awake  Kick  Kick  Kick  Last kick |