Watchdog Timer

# Introduction

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

**wdreset.c**

The callout\_init()function initiates the callout object. os\_sem\_init initiates the semaphore with a default value of 0. Watchdog\_init initializes the watchdog with the specified timeout in microseconds and the watchdog timer NULL called for system reset. The watchdog\_start() function will start the watchdog counter.

|  |
| --- |
| callout\_init(&wakeup, wakeup\_callback);  os\_sem\_init(&wakeup\_event, 0);  watchdog\_init(WATCHDOG\_TIMEOUT, NULL);  watchdog\_start(); |

callout\_init\_soft()initializes callout object with a soft deadline. 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.

|  |
| --- |
| os\_suspend\_enable();  callout\_schedule(&wakeup, SYSTIME\_SEC(2));  os\_printf("Sleeping\n");  os\_sem\_wait(&wakeup\_event);  os\_printf("Awake\n");  os\_suspend\_disable();  os\_usleep(SYSTIME\_SEC(4)); |

This call back function is to wake up when the semaphore returns.

|  |
| --- |
| static void  wakeup\_callback(struct callout \*c)  {  os\_sem\_post(&wakeup\_event);  } |

This callback function is to trigger 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(), it is no longer seen and thus the watchdog timer expires.

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

## Running the Application

Program wdreset.elf(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.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y/pc\_tools/Download\_Tool/doc*).

**Note**:

1. x and y refer to the SDK release version. For example: sdk\_2.4/doc.
2. Post 30th call of the watchdog\_kick()function:
   1. Prog RAM: watchdog timer expires
   2. Prog Flash: watchdog timer runs in an infinite loop

For details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf.

## Expected Output

|  |
| --- |
| UART:NWWWWAE  4 DWT comparators, range 0x8000  Build $Id: git-7e2fd6a94 $  hio.baudrate=115200  flash: Gordon ready!  UART:NWWWAEBuild $Id: git-65f6c1f46 $  $App:git-e3ccdc7a  SDK Ver: sdk\_2.4  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 |