Tutorial

Starting a Project

For the DA1468x SoC

Abstract

This tutorial should be used as a reference guide to gain a deeper understanding of the DA1468x family of devices. As such, it covers a broad range of topics including a brief discussion on the software tools and the usage of the most common hardware blocks of the device. Furthermore, it covers a number of sections containing in depth software analysis of the fundamental code found on every sample application.

Contents

Figures

Tables

Terms and Definitions

AHB	AMBA High Speed Bus Clock
APB	AMBA Peripheral Bus Clock
BLE	Bluetooth Low Energy
LP	Low Power
ms	Milliseconds
OS	Operating System
ProDev	Pro Development Kit
USB	Universal Serial Bus

References

[1] UM-B-044, DA1468x Software Platform Reference, User Manual, Dialog Semiconductor.

[2] DA14680-01 DS v2.1 Datasheet, Low Power Bluetooth Smart 4.2 SoC with Flash, Dialog Semiconductor.

1 Introduction

1.1 Before We Start

Before we start you need to:

- Install the latest SmartSnippets Studio
- Download the latest SDK for the DA1468x platforms

These can be downloaded from the Dialog Semiconductor support portal.

For this tutorial, either a Pro or Basic Development kit is required.

The key goals of this tutorial are to:

- Start working with a customized project
- Analyze the key points of the code structure
- Prepare both hardware and software to run a demonstration code example

2 Creating a New Project

This section describes how to import a project containing source code and start working with a custom one. There are two ways to start working with a custom project, either import an existing sample design and make all the necessary changes after copying it or build a new one from the scratch. Since the latter is time consuming and out of the scope of this tutorial, only the first process is demonstrated.

2.1 Importing a Project

To import a project:

1. On the SmartSnippets Welcome page, click **Browse** (*1.1*) in the **SOFTWARE RESOURCES** section.

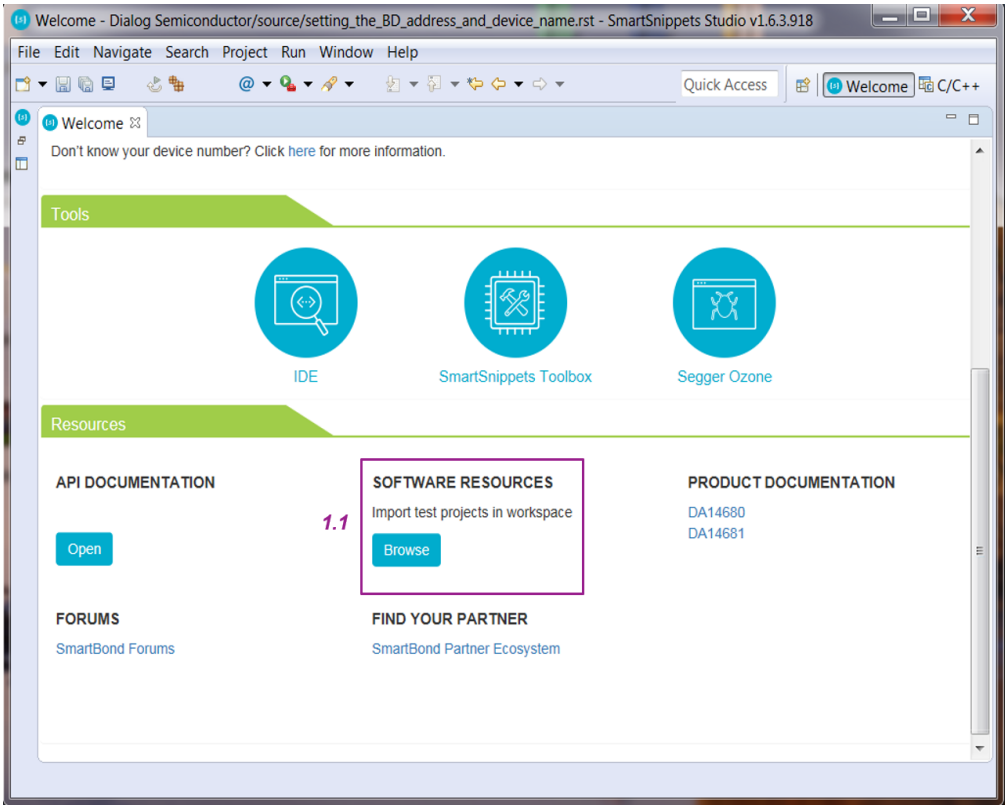

Figure 1: Open Project Browser

2. In the pop-up window, click **OK** (*2.1*) as your current workspace folder should be automatically selected. If this is not the case, you must explicitly select it.

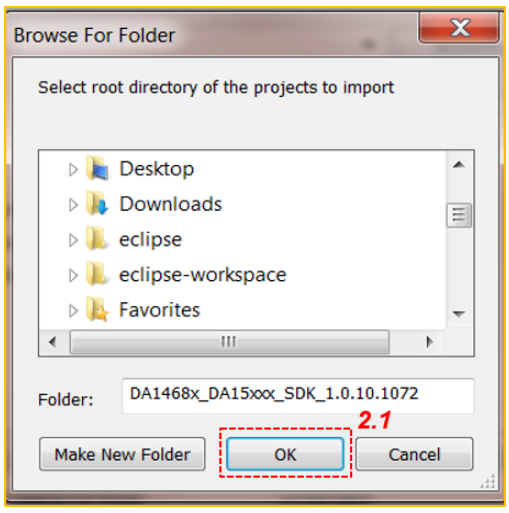

Figure 2: Select SDK Folder

3. The final step is to select the preferred project(s) to import. By default all projects are selected. The simplest way to continue is to:
    1. Click **Deselect All** (*3.1*).
    2. Select the desired projects by clicking on the respective **tick boxes** (*3.2*).

3. Click **Finish** (*3.3*).

Now you are ready to start working with the project.

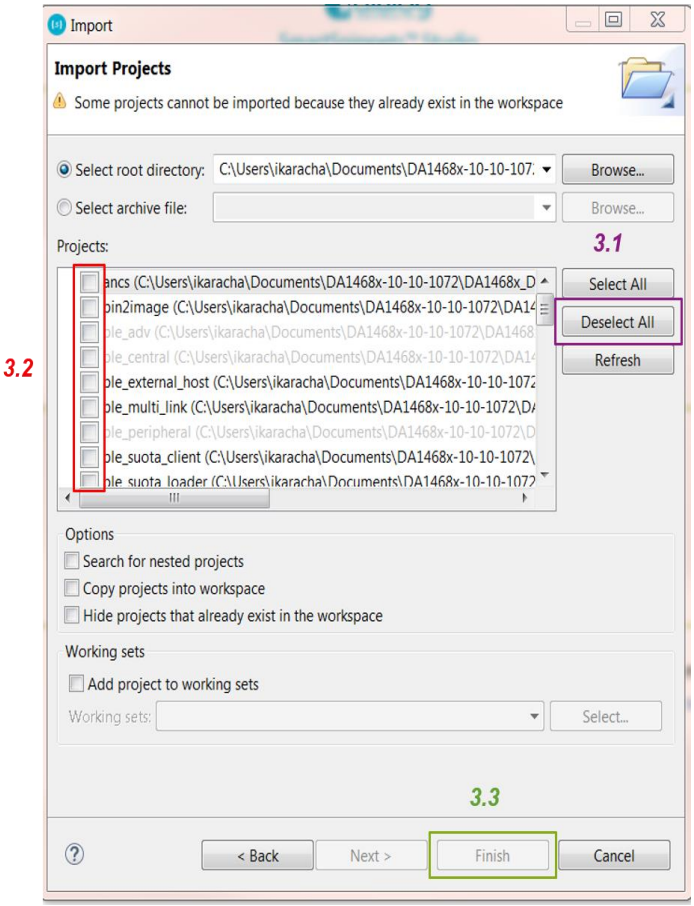

Figure 3: Project Selection

2.2 Configuring the Workspace

The **SEGGER J-Link** drivers must be correctly installed in SmartSnippets ™ Studio. To check this:

1. Click **Window > Preferences**.

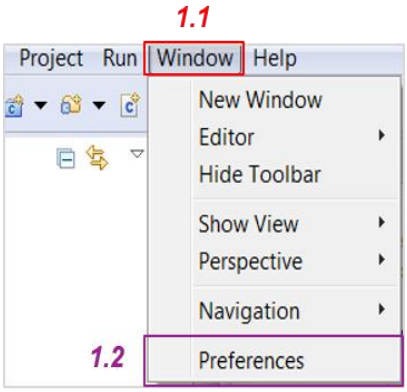

Figure 4: Configure SEGGER J-Link Tool Chain

2. In the displayed window, go to **Run/Debug > SEGGER J-Link** and, in the **Folder** section, click **Browse...**. Browse for the J-Link drivers (normally found under *C:\Program Files (x86)*).

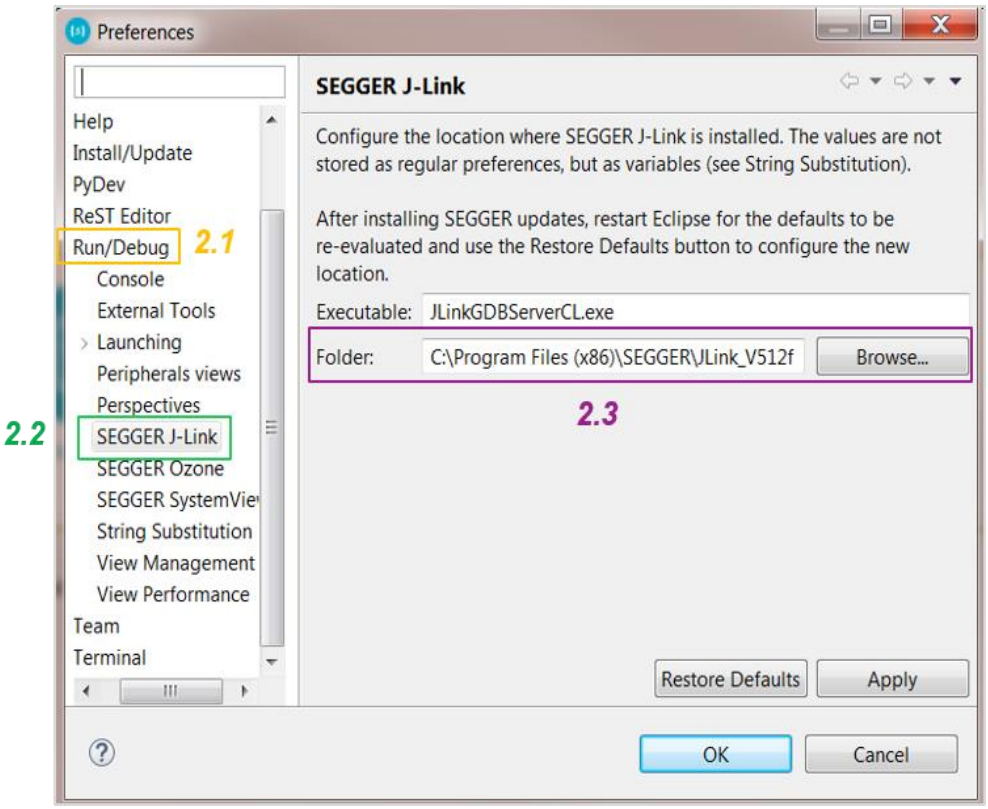

Figure 5: Second Step to Configure the Workspace

The same procedure should be carried out for **SEGGER Ozone** and **SystemView**. If one of these tools is not already installed then you need to download it from SEGGER's official web site.

2.3 Copying a Demo Project

As mentioned in previous section, there are two phases to start working with a custom project. The first stage is to import a demo project from Dialog's SDK and the second is to copy it and make all the required changes. The recommended way to copy sample code is:

1. Right-click on the imported sample code folder and select **Copy**.

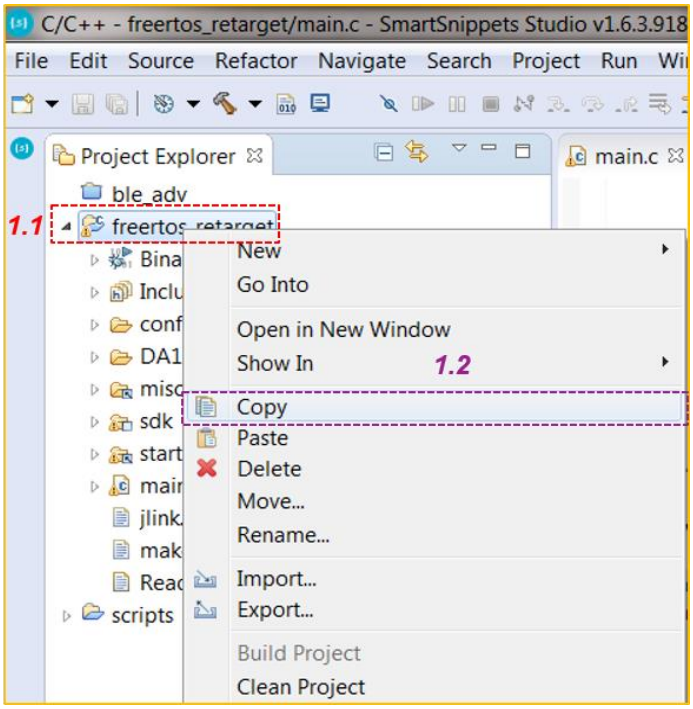

Figure 6: First Step to Copy a Project

2. Right-click on a blank area (in the **Project Explorer**) and select **Paste**. In the **Copy Project** window, enter a **Project name** (*2.1*) and uncheck **Use default location** (*2.2*). **Browse...** for a location to store the new project (*2.3*).

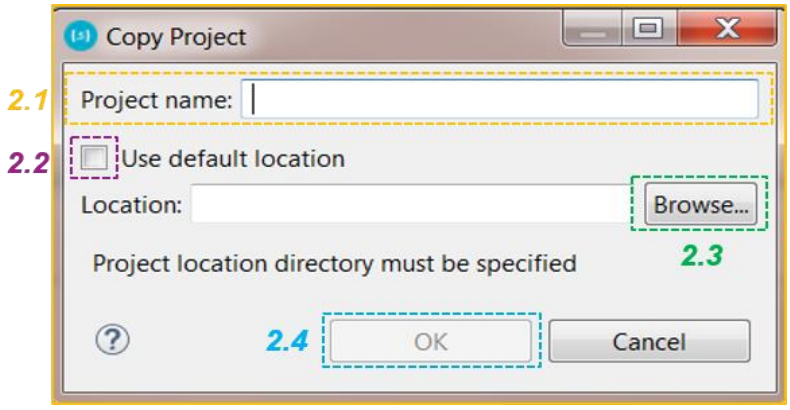

Figure 7: Second Step to Copy a Project

3. In the **Browse For Folder** window, select the location where the new project will be stored (*3.1*). It is recommended to select one of the folders under which the various sample codes are stored in the SDK (for example, <sdk_folder>/projects/dk_apps/<imported_project_location>). Create a new folder by clicking on **Make New Folder** (*3.2*) and entering a folder name (*3.3*). Click **OK** (*3.4*).

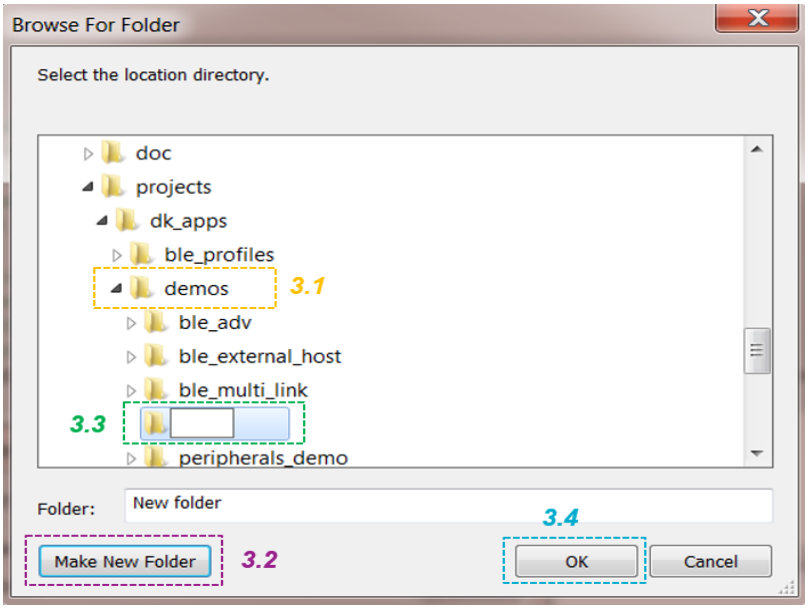

Figure 8: Third Step to Copy a Project

4. Click **OK** in the **Copy Project** window (2.4). The newly copied project will be displayed in **Project Explorer** below the imported project.

Note: The new project must remain at the same hierarchical depth as the source project as some of the environment variables rely on the relative path. If the user fails to keep the depth consistent, the project will fail to compile.

2.4 Adding a New Folder/File to the Project

1. To add a new folder to your project, select **File > New > Folder**.

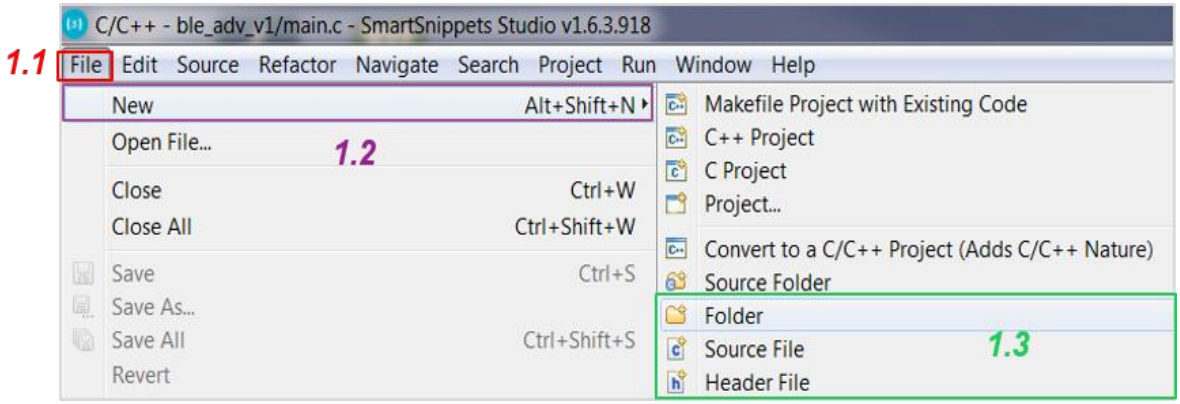

Figure 9: Select New Folder Menu

2. In the **New Folder** window, select the path under which the new folder will be added (*2.1*), enter the **Folder name** (*2.2*), and click **Finish** (*2.3*). The new folder will be displayed.

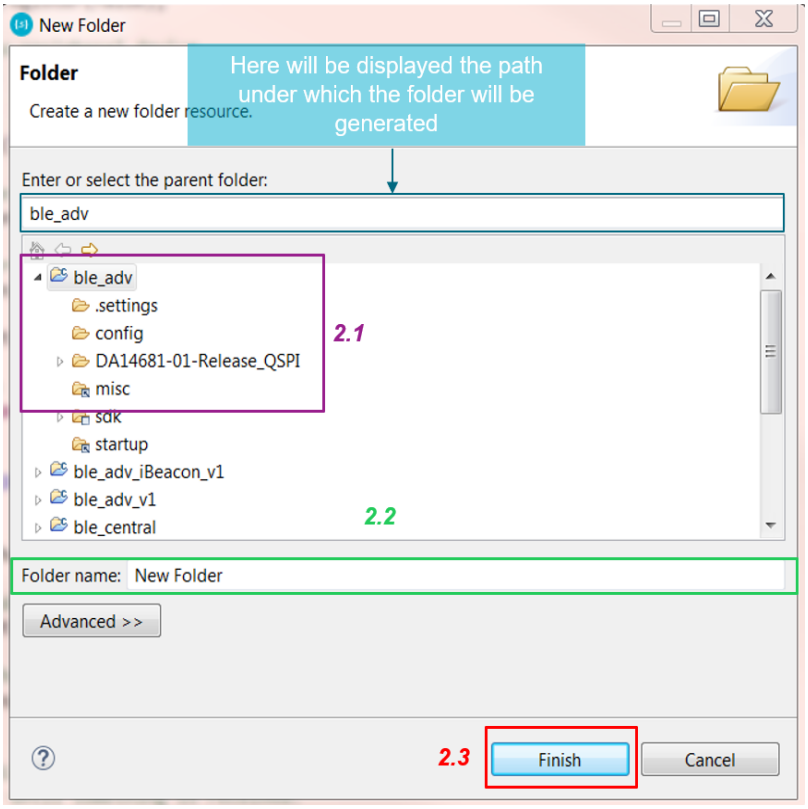

Figure 10: Select Parent Folder, Input New Name

3. Use the same procedure to define new **source** and **header** files (at step 1.3 select either **Source File** or **Header File** respectively, instead of selecting **Folder**). In this case, the preferred names (step 2.2) should end with a **.c** or **.h** suffix respectively (for example, source.c or header.h).

4. Every newly created folder must be placed inside the project's path, in a folder that is in the compiler's include path. To do this, select Project > Properties > C/C++ Build > Settings > Tool Settings > Cross ARM C Compiler > Includes and click on the **Add...** icon (*4.6*).

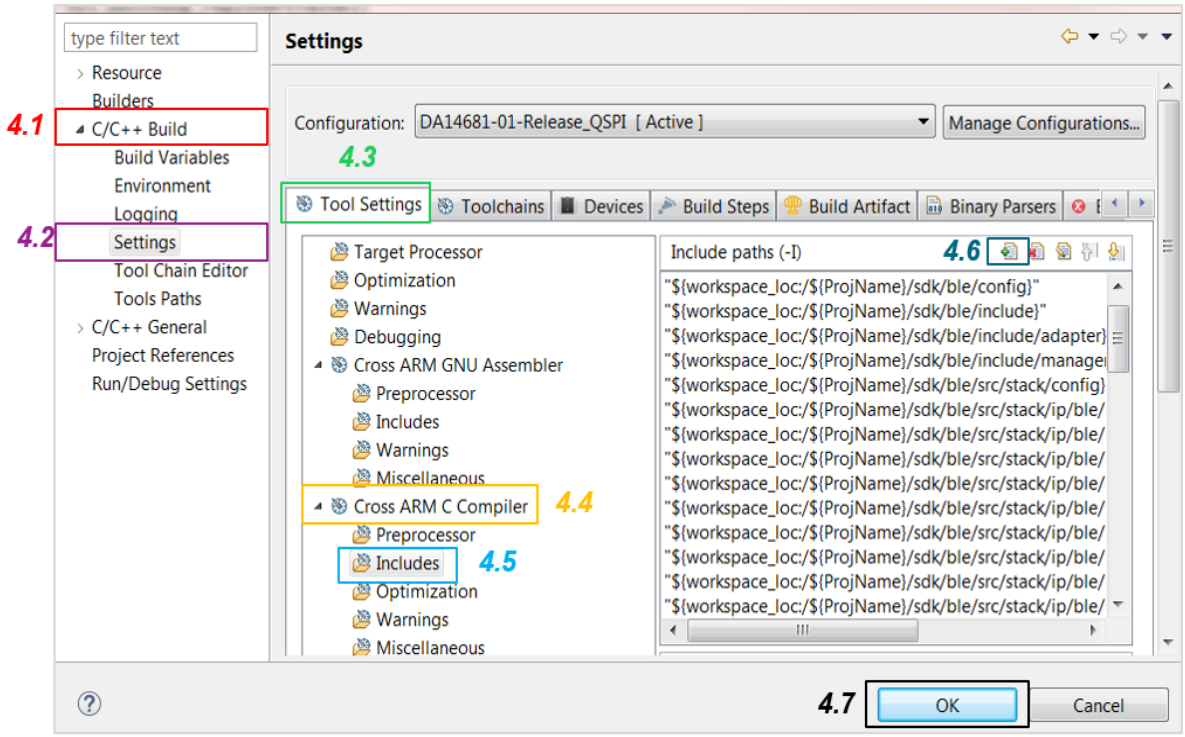

Figure 11: Add the New Folder to the Include Search Path

5. In the **Add directory path** window, click **Workspace...** (*5.1*) and specify the path of the newly created folder/file. It will be displayed in the **Directory** section. Click **OK** (*5.2*) and then click **OK** in the previously opened window (*4.7*).

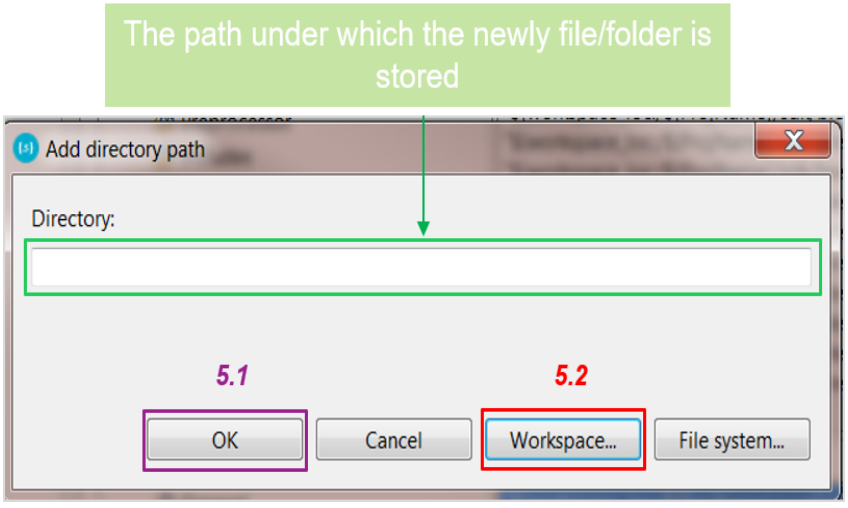

Figure 12: Input New Folder Name

3 Analyzing the Key Points of a Sample Code

This section covers the fundamental code structure found in every demonstration application included in Dialog's SDK. All applications are implemented as FreeRTOS tasks that are created under the system_init() function.

3.1 Clock and Power Manager

The clock manager is part of the clock and power manager (CPM) and it:

- Controls the system level clocks. The divider of each hardware resource is responsible to control the clock settings for the resource. In this context, the CPM controls the system clock which is illustrated with green line (sys_clk) in Figure 13, the AHB and APB clocks which are the blue lines at the top of the figure, and the low power clock which is the black line (lp_clk).
- Handles requests to switch to another clock configuration (cm_sys_clk_set()). A request may be denied if this switch affects a hardware resource that is active (for example, a hardware timer).
- Offers the application tasks the ability to switch to another clock configuration during runtime, if possible. The low power clock cannot be changed during runtime.

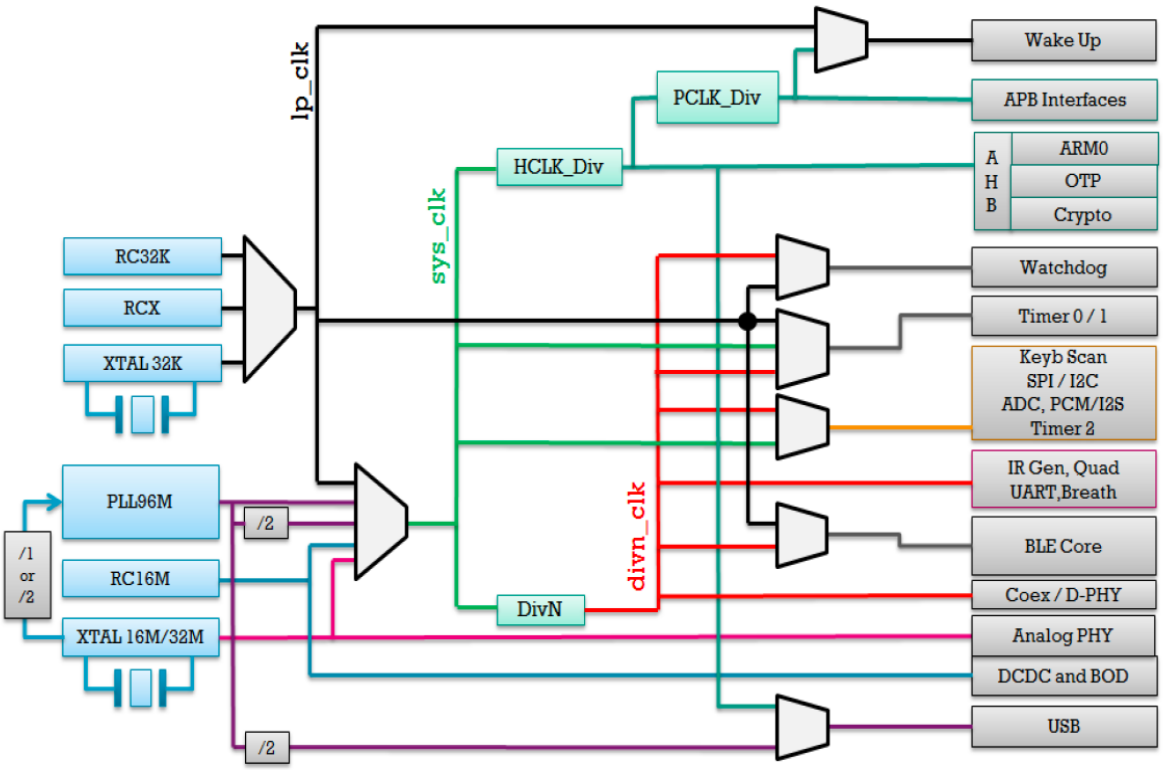

Figure 13: The Clock Tree Diagram for the DA1468x Family of Devices

SmartBond ™ DA1468x SDK abstracts the complexity of the clock tree from the applications by offering a unified API for clock control. The details of the API are highlighted in Table 1

Table 1: Dialog's API for Clock Control

Function name	Description
bool cm_sys_clk_set(sys_clk_t type)	Sets the system clock. The available options are: RC16, XTAL16M (or XTAL32M), PLL48, and PLL96. The low power clock cannot be set as the system clock.
bool cm_cpu_clk_set(cpu_clk_t clk)	Sets the system clock and the AHB divisor such that the requested clock frequency is achieved.
void cm_apb_set_clock_divider(apb_div_t div)	Sets the clock divisor for the APB clock. The actual frequency depends on the system clock used.
bool cm_ahb_set_clock_divider(ahb_div_t div)	Sets the clock divisor for the AHB clock. The actual frequency depends on the system clock used.
_get and _fromISR	Variants of the above _set functions.
void cm_lp_clk_init(void)	Initializes the Low Power clock.
bool cm_lp_clk_is_avail(void)	Checks if the Low Power clock is available.
void cm_clk_init_low_level(void)	Executes clock initialization after power-up.
void cm_sys_clk_init(sys_clk_t type)	Executes clock initialization after the OS has started.

3.2 The Key Points of the Main Function

The main() function is the basic routine of every source code and is responsible for the following actions:

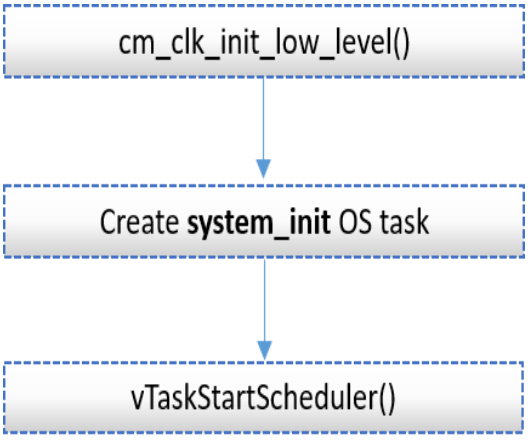

Figure 14: SW FSM of main() Function

1. The cm_clk_init_low_level function must be called before any other configuration related to system clocks. It is the lowest level function. More specifically, it switches to the internal 16MHz RC oscillator (**RC16M**), restarts the external 16 MHz crystal **(XTAL16M)** and may wait for it to settle. It also sets the **DIVN** clock divider which provides the 16 MHz clock source to various hardware blocks of the device, regardless of the system clock source. It then sets up the low power clock, according to the value of the dg_configUSE_LP_CLK macro in the config/custom_config_qspi.h header file. This function must be called only once, before the freeRTOS scheduler is started.

Note: One case where the system does not wait for the settling of the XTAL16M, is when both the external 32kHz crystal is selected as low power clock (LP_CLK_32768) and it has explicitly defined not to wait for the settling of the crystal, that is, pm_set_wakeup_mode(false). The hardware blocks that require the XTAL16M to properly work are the BLE, USB and the UART peripheral.

2. The next step is to create the task named system_init that is responsible for initializing the system. In this FreeRTOS task all the application tasks should be declared as well.
3. After creating all the required FreeRTOS tasks, it's time to start running the scheduler which is responsible for controlling and executing all the previous defined tasks, according to their priorities.

3.3 The Key Points of the System Initialization Function

1. All the system clocks should be initialised according to the target application needs. In specific, the system clock, low power clock source, clock divider for the AMBA peripheral bus clock (**APB**) and AMBA high speed bus (**AHB**) are set.

Code Snippet:

```
/*
 * Prepare clocks. Note: cm_cpu_clk_set() and cm_sys_clk_set() can only be called from a
 * task since they will suspend the task until the XTAL16M has settled and the PLL maybe
 * locked.
 */
 cm_sys_clk_init(sysclk_XTAL16M); // Set the system clock
 cm_apb_set_clock_divider(apb_div1); // Set divider for the APB bus
 cm_ahb_set_clock_divider(ahb_div1); // Set divider for the AHB bus
 cm_lp_clk_init(); // Set the clock source of the low power clock
```

Note: The function which sets the low power clock does not have any input parameters. The desired clock source must be defined using the macro dg_configUSE_LP_CLK, preferably set in the /config/custom_config_qspi.h header file. The valid values this macro can accept is: LP_CLK_RCX (internal RC oscillator), LP_CLK_32768 (external 32 kHz crystal) or LP_LCK_32000 (external clock pulses). By default all code examples found in Dialog's SDK use the external crystal 32 kHz as the LP clock, that is, #define dg_configUSE_LP_CLK LP_CLK_32768. Table 2 contains the system clock source enumerated type values.

Table 2: System Clock Types

Enumeration name	Value	Description
sysclk_RC16	0	Sets the internal RC oscillator of 16 MHz as the source clock.
sysclk_XTAL16M	1	Sets the external crystal of 16 MHz as the source clock.
sysclk_XTAL32M	2	Sets the external crystal of 32 MHz as the source clock.
sysclk_PLL48	3	Sets the PLL block at 48 MHz as the source clock.

sysclk_PLL96	6	Sets the PLL block at 96 MHz as the source clock.
sysclk_LP	255	Sets the low power clock as the source clock.

Note: If the external 32kHz crystal is selected as low power clock, the system is prevented from entering sleep for approximately 8 seconds after a HW reset (cold boot). This delay ensures the external crystal is settled before using it as a low power clock for the system. After that time delay, the device can enter a sleep mode. The low power clock (LP) is used during the device sleep. Also note that sysclk_LP cannot be used as system clock source.

2. The next step is the initialization of the device's peripherals used in the project, for example, the pins multiplexing for the UART interface. The state configurations are not retained during sleep and have to be restored at wake up. All device configurations should be placed in prvSetupHardware function. In this function pm_system_init routine is called as well. This function performs the actual device's pin initialization after a power-up and a wake-up cycle.
3. The next step is the selection of the sleep mode. Please note that the recommended sleep mode for all applications is the **extended sleep mode.**

Code Snippet:

```
/* Set the desired sleep mode */
pm_set_wakeup_mode(true); // If the input parameter is set to "true" then the device waits until
//XTAL16M is settled.
pm_set_sleep_mode(pm_mode_extended_sleep);
```

Note: After wake-up, the system runs using the internal 16 MHz (RC16) clock oscillator. The clock manager (CM) switches back to the last system clock configuration (before sleep) and the FreeRTOS resumes its tasks either immediately or after the external 16MHz crystal (XTAL16M) has settled. This procedure is transparent to the application tasks. The CPM unblocks any task that has been blocked, waiting for the high precision clock (external crystal 16 MHz). The hardware blocks of the device that require the XTAL16M to properly work are the Bluetooth (BLE), USB and the UART peripheral.

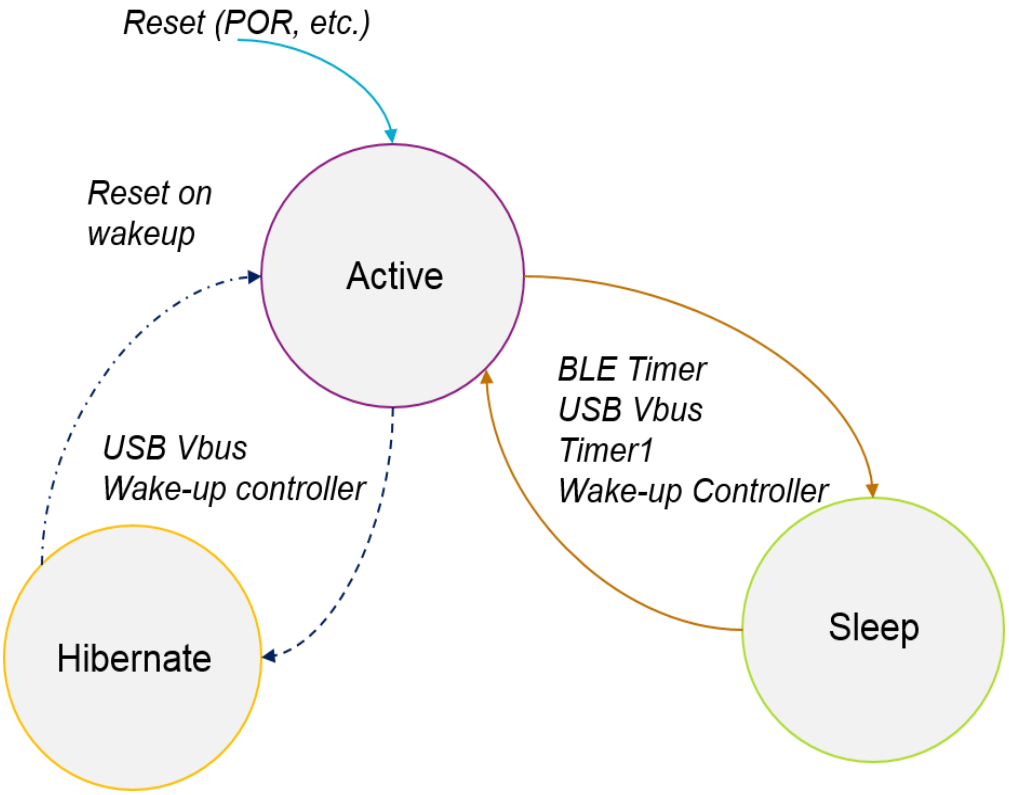

Figure 15: Possible Device's States

4. After all the mandatory configurations, all the application tasks that will be executed by the FreeRTOS scheduler should be declared.

Note: The last code line of system_init task is OS_TASK_DELETE(OS_GET_CURRENT_TASK()) for deleting the task itself. This is done in order to release valuable system resources. After all, the initialization of the system only needs to be done once at beginning of the device's start.

4 Running The Demonstration Example

This section describes the steps required to prepare the Pro DevKit and other tools to successfully run an example code that is based on **freertos_retarget** sample code found in the SDK (demonstrates using the UART functionality).

To make things more interesting, let's give some extra functionality to the device making the LED D2 on Pro DevKit to toggle after pressing a push button. The Wakeup controller can be programmed to wake up the DA1468x family of devices from a power-down mode after a pre-programmed number of GPIO events. In summary, this module consists of an event counter and a timer that counts every 1 ms. If the event counter reaches its pre-defined value, the counter is reset and an interrupt is generated. Each of the GPIO inputs can be selected to generate an interrupt event. As far as the timer is concerned, this hardware block can add a delay until an event (**Key Hit**) is being considered valid. This feature works as follow: upon an event the timer starts counting-down from its pre-defined value and, if it reaches zero and the **Key hit** is still valid, then the event counter is incremented. This

feature is useful when using mechanical parts to trigger the controller by eliminating debouncing issues.

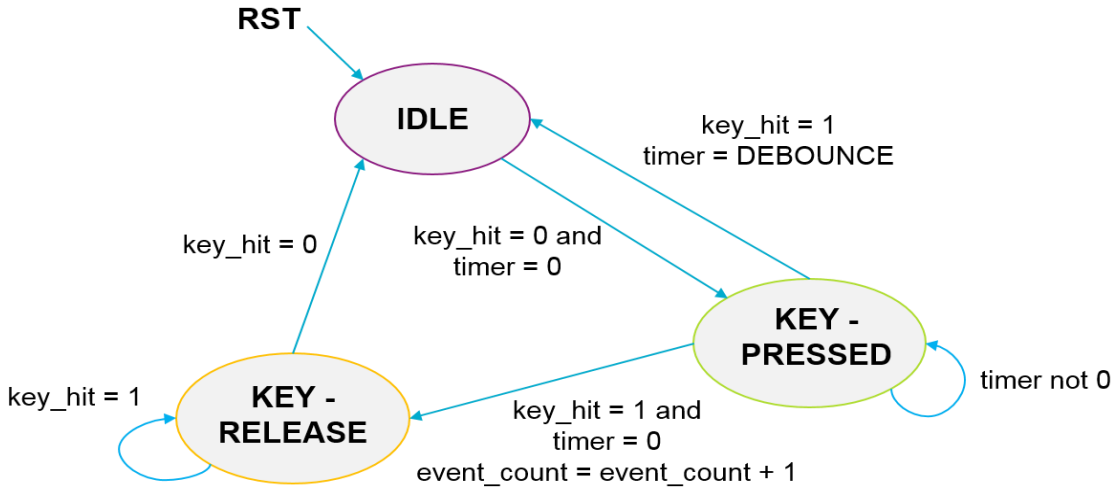

Figure 16: Event Counter State Machine for the Wake-Up Interrupt Generator

Note: The event counter is edge sensitive. After detecting an active edge, a reverse edge must be detected first before it goes back to the IDLE state and from there starts waiting for a new active edge.

Note: The Wakeup controller can be used even when the system is set to always active.

There are two main methods to verify the correct behavior of the demonstrated code. The first method is to use a Serial Terminal and the second is to use the SmartSnippets Toolbox.

4.1 Verifying with a Serial Terminal

1. Establish a connection between the target device and your PC through the **USB2(DBG)** port of the motherboard. This port is used both for powering and communicating to the DA1468x SoC. For this tutorial a Pro DevKit is used. Ensure the jumper settings are set as displayed in the following figure:

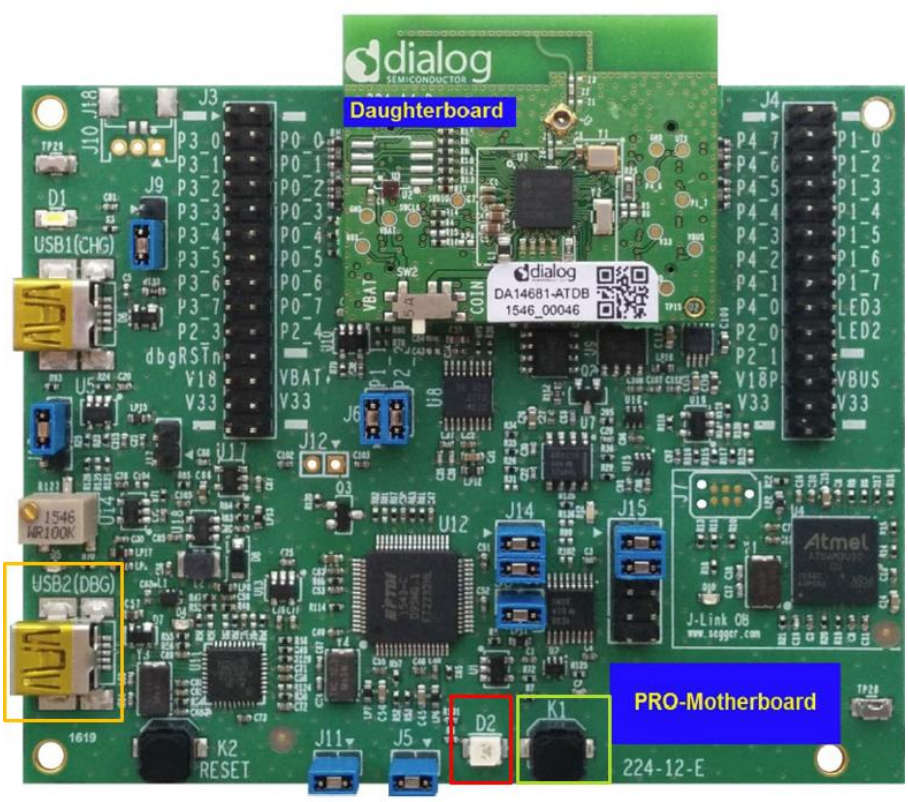

Figure 17: The ProDev Kit (Motherboard and Daughterboard)

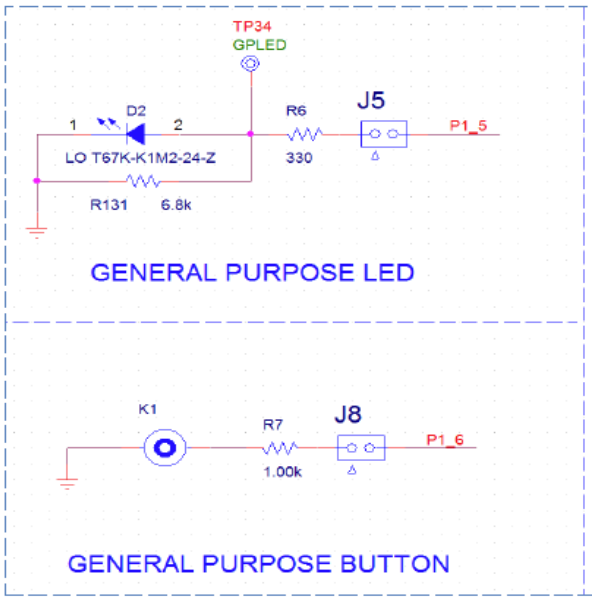

Figure 18: General Purpose LED and BUTTON Schematics

Table 3: DA1468x Pin Assignment of the General Purpose LED and BUTTON

DA1468x Pin Name	Signal Name	Multiplexed with Functions
P1_5	RTS	D2 LED, through J5
P1_6	CTS	K1 push button, through jumper J8

2. Import and then make a copy of the **freertos_retarget** sample code found in the SDK of the DA1468x family of devices.

Note: It is essential to import the folder named scripts to perform various operations (including building, debugging, and downloading)

3. In the target application, add/modify all the required code blocks as illustrated in the Code Overview section.
4. Build the project either in **Debug_QSPI** or **Release_QSPI** mode and burn the generated image to the chip.
5. Press the **K2** button on Pro DevKit to start the chip executing its firmware.
6. Select a serial console and open it. In this tutorial we have selected **Tera Term** which is a free and easy-to-use serial terminal. Follow the steps in the following figures to successfully establish a connection via the USB port. If you connect the device it is automatically displayed as an option (*2*).

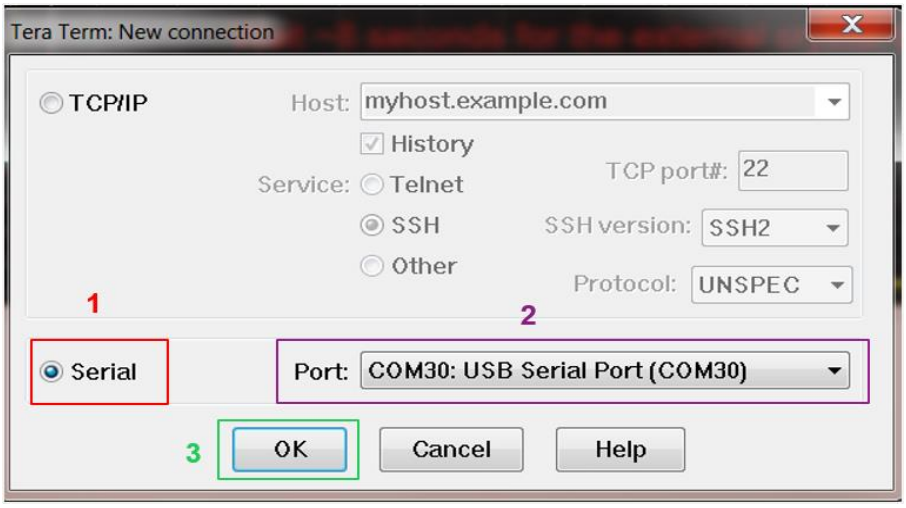

Figure 19: Select the Serial Port to which the Development Kit is Connected

In the displayed window, select the UART parameters:

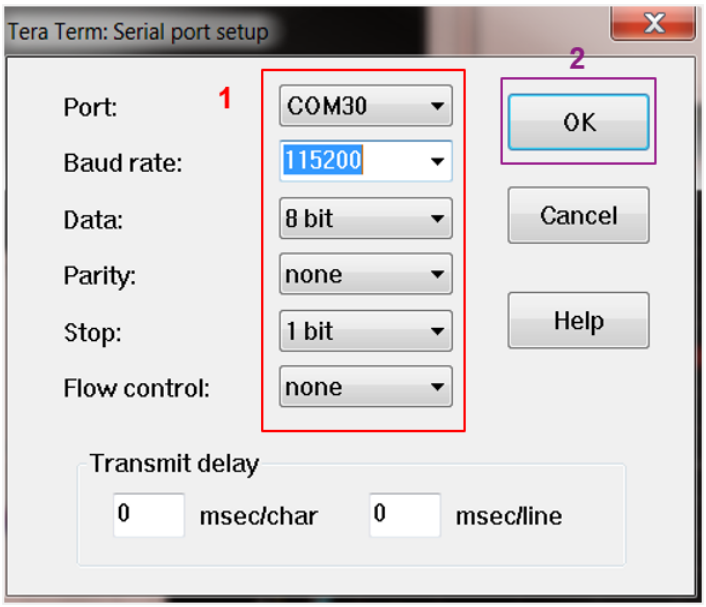

Figure 20: Configure the Serial Port with the Correct Settings

After successfully setting the serial port you should see the special character '#' displayed on the console.

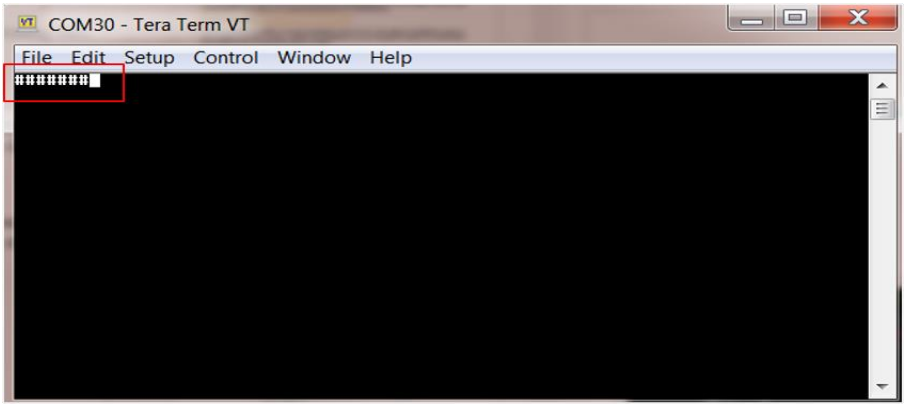

Figure 21: The Special Character '#' is Displayed on the Screen

7. Press and release the **K1** button on Pro DevKit three times. LED **D2** on Pro DevKit should be turned on. For as long as LED D2 is active, the device is not allowed to enter sleep. Press and release the K1 button on Pro DevKit three times again. LED D2 should be turned off and the device should enter sleep.

Note: The number of events required for the led to toggle is configured through hw_wkup_set_counter_threshold API in init_wkup routine.

4.2 Verifying with SmartSnippets Toolbox

1. With the system up and running, open the SmartSnippets Toolbox and execute the following steps:
    a. (Optional) Select **New** to create a new project (1). In the **New Project** window, enter a name for the project (2). This step is optional if a project has already been created.
    b. Choose an available project (4).
    c. Choose a communication interface (3) and a port (5).
    d. Select the family of devices to use (6).
    e. Open the selected project (7).

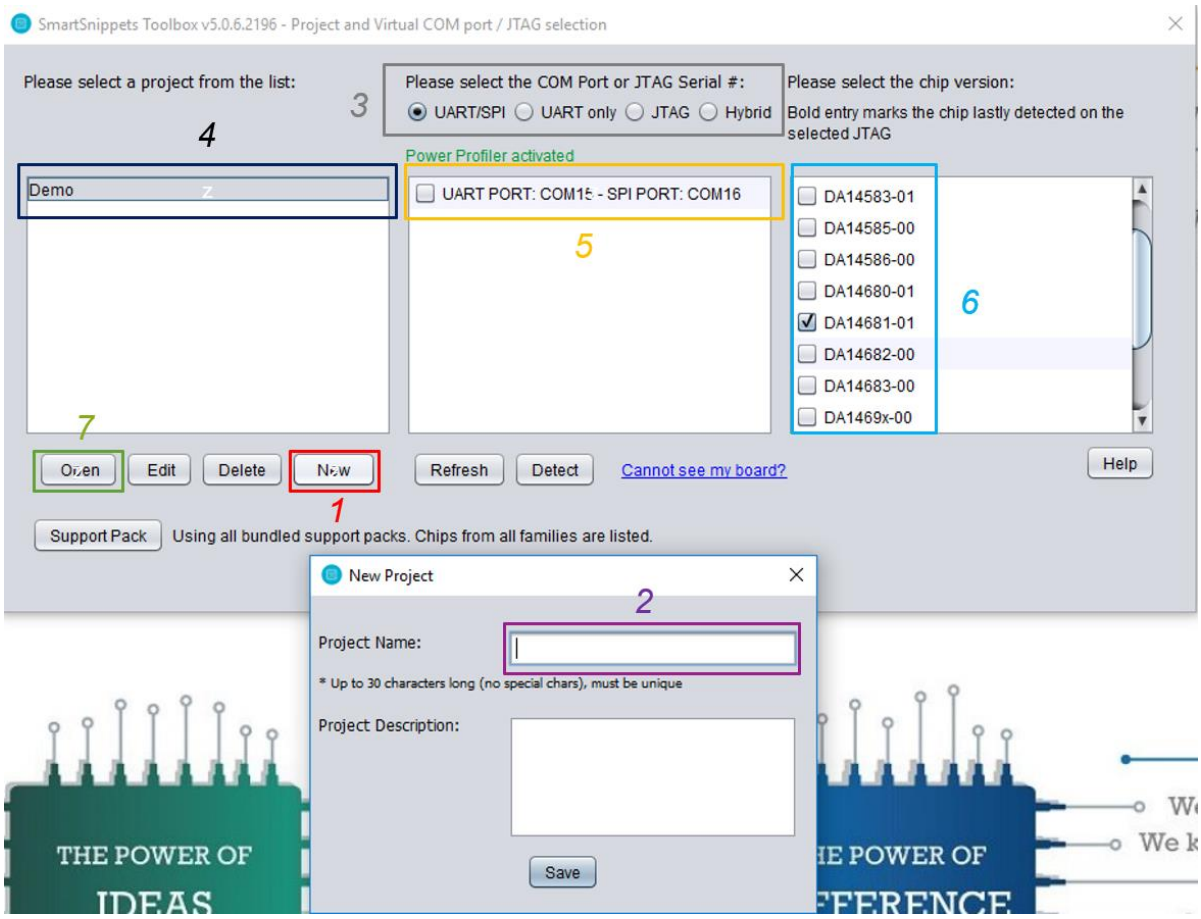

Figure 22: Opening a Project in the SmartSnippets Toolbox

2. Start power profile monitoring:

a. Switch to the **Power Profiler** window (1).

b. Initialize Power Profiler (2).

c. Start Power Profiler (3).

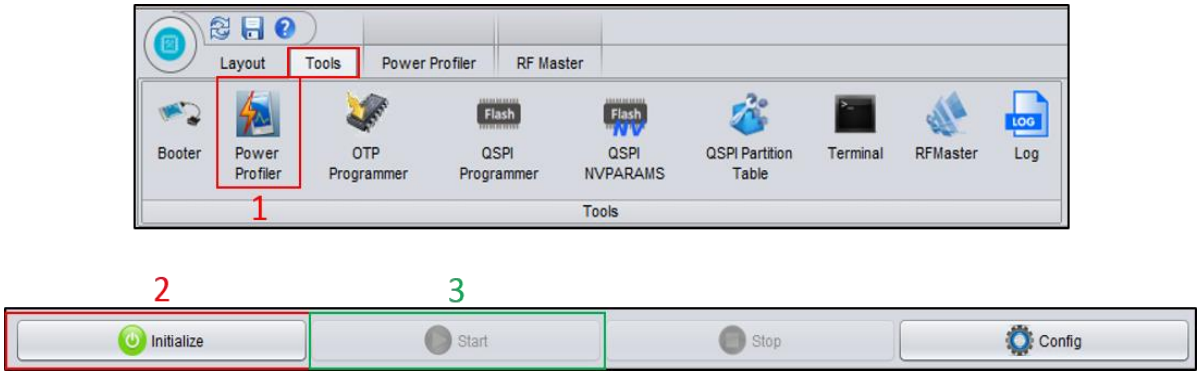

Figure 23: Initializing Power Profiler

3. Press the **K2** button on Pro Dev Kit to reset the device. It should enter sleep approximately 8 seconds after a power-up event.

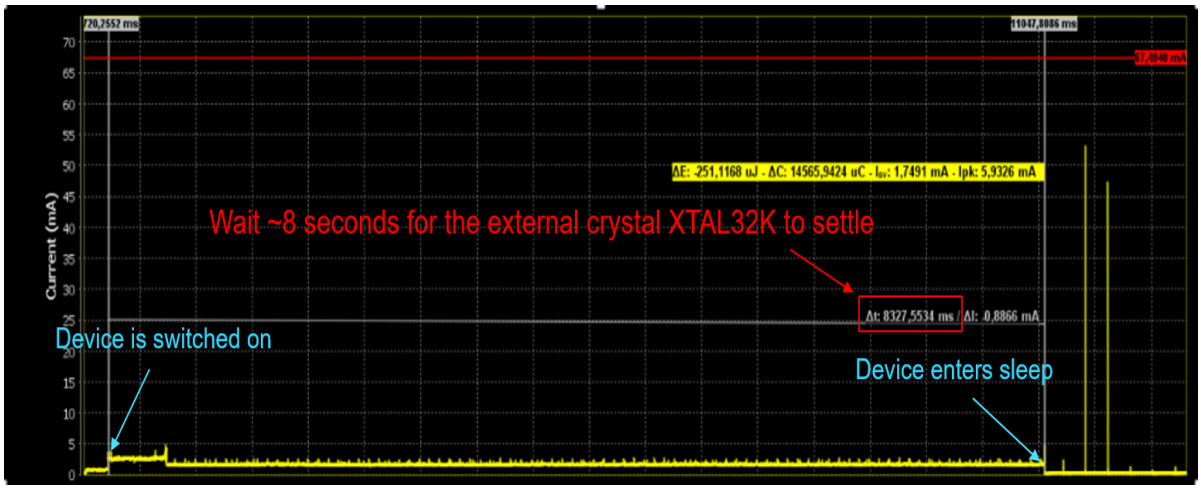

Figure 24: Verifying the Correct Functionality of the Device before and after Entering Sleep

After the passage of 8 seconds the device should wake up every 200ms (LED D2 is turned off).

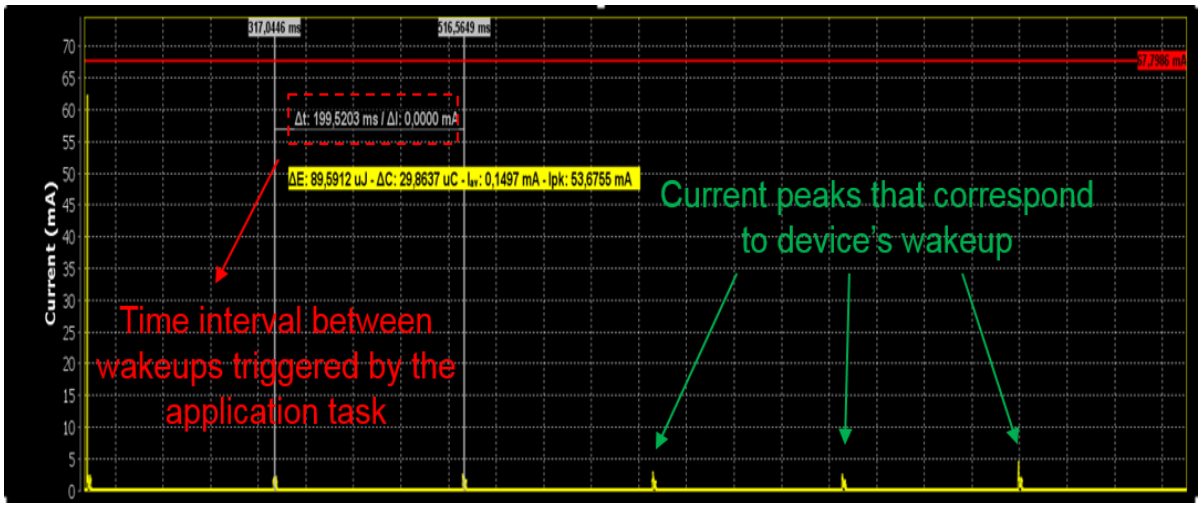

Figure 25: Verifying the Correct Functionality of the Device between Wake-Ups

Press and release the **K1** button on Pro DevKit three times. Verify the increased power consumption while the device is in active mode (LED D2 on).

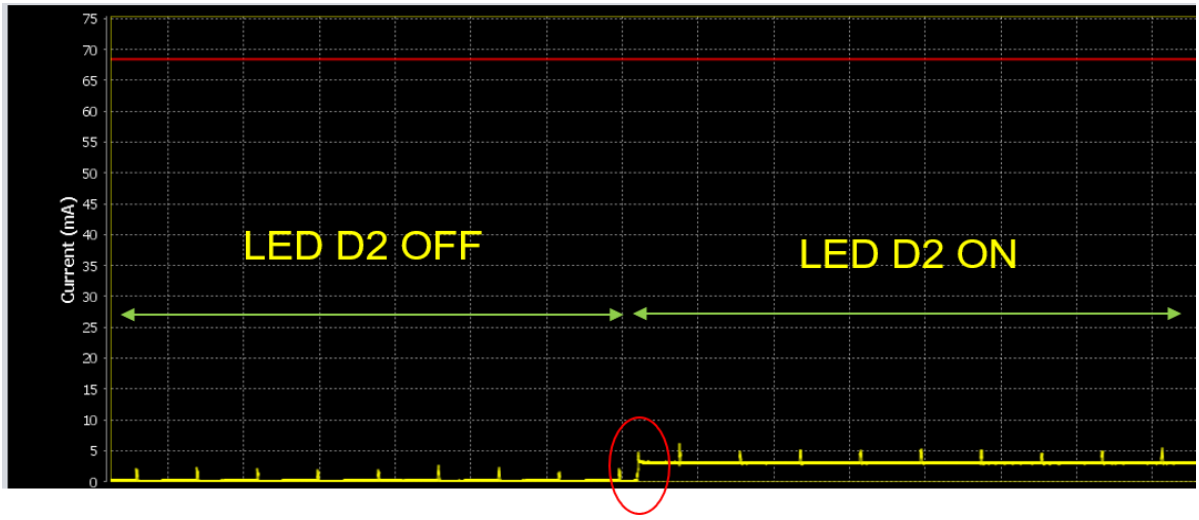

Figure 26: Verifying the Power Consumption while in Active and Sleep Mode

5 Code Overview

This section contains the code blocks needed to successfully execute this tutorial.

5.1 Header Files

In main.c, add the following header file:

```
/* For the Wakeup Timer */

#include "hw_wkup.h"
```

5.2 System Init Code

In main.c, replace system_init() with the following code:

```
static void system_init( void *pvParameters )
{
        OS_TASK task_h = NULL;

#if defined CONFIG_RETARGET
        extern void retarget_init(void);
#endif

        /* Prepare clocks. Note: cm_cpu_clk_set() and cm_sys_clk_set() can only be called from a
         * task since they will suspend the task until the XTAL16M has settled and the PLL maybe
```

```
         * is locked.
         */
        cm_sys_clk_init(sysclk_XTAL16M);
        cm_apb_set_clock_divider(apb_div1);
        cm_ahb_set_clock_divider(ahb_div1);
        cm_lp_clk_init();

        /* Prepare the hardware to run this demo. */
        prvSetupHardware();

        /* init resources */
        resource_init();

#if defined CONFIG_RETARGET
        retarget_init();
#endif

        /* Set the desired sleep mode. */
        pm_set_wakeup_mode(true);
        pm_set_sleep_mode(pm_mode_extended_sleep);

        /* Start main task here (text menu available via UART1 to control application) */
        OS_TASK_CREATE( "Template",                 /* The text name assigned to the task; for
                                                    debug only, not used by the kernel. */
                    prvTemplateTask,                /* The function that implements the task. */
                    NULL,                           /* The parameter passed to the task. */
                    200 * OS_STACK_WORD_SIZE,       /* The number of bytes to allocate to the
                                                    stack of the task. */
                    mainTEMPLATE_TASK_PRIORITY,     /* The priority assigned to the task. */
                    task_h );                       /* The task handle. */
        OS_ASSERT(task_h);

        /* The work of the SysInit task is done */
        OS_TASK_DELETE( xHandle );
}
```

5.3 Wake-Up Timer Code

In main.c, after system_init(), add the following code for handling external events via the wake-up controller:

```
PRIVILEGED_DATA volatile bool pin_status_flag = 0;

/* Callback function to be called after an interrupt is generated, that is, when event counter reaches
configured value. */
void wkup_cb(void)
{
      /* This function must be called by any user-specified interrupt callback, to clear the interrupt flag.
*/
        hw_wkup_reset_interrupt();
```

```
        /* Toggle the LED D2 on the Pro DevKit */
        hw_gpio_toggle(HW_GPIO_PORT_1, HW_GPIO_PIN_5);

        /* Toggle the value of the flag indicating the status of the LED before entering sleep */
        pin_status_flag ^= 1;

}

/* Function which makes all the necessary initializations for the wake-up controller. */
static void init_wkup(void)
{
        /* This function must be called first and is responsible for the initialization of the hardware block
*/
        hw_wkup_init(NULL);

        /*
         * Configure the pin(s) that can trigger the device to wake-up while in sleep mode. The last input
parameter
         * determines the triggering edge of the pulse (event)
         */
        hw_wkup_configure_pin(HW_GPIO_PORT_1, HW_GPIO_PIN_6, true,
HW_WKUP_PIN_STATE_LOW);

        /*
         * This function defines a delay between the time a trigger event will be presented and the time
the controller
         * will take this event into consideration.
         * Setting debounce time to 0 disables hardware debouncing. Maximum debounce time is 63
ms.
         */
        hw_wkup_set_debounce_time(10);

#if dg_configBLACK_ORCA_IC_REV == BLACK_ORCA_IC_REV_A // Check if the chip is either
//DA14680 or 81
        /*
         * Set threshold for event counter. Interrupt is generated after the event counter reaches the
configured value.
         * This function is only supported in DA14680/1 chips.
         */
        hw_wkup_set_counter_threshold(3);
#endif
        /* Register interrupt handler. */
        hw_wkup_register_interrupt(wkup_cb, 1);
}
```

Note: If the variable named pin_status_flag is not used, LED D2 on Pro DevKit will lose its state after entering sleep. The default state for all pins when entering sleep is input pull-down.

5.4 Hardware Initialization

In main.c, replace both periph_init() and prvSetupHardware() with the following code to configure pins after a power-up/wake-up cycle. Please note that every time the system enters sleep, it loses all its pin configurations.

```
/**
 * @brief Initialize the peripherals domain after power-up.
 *
 */
static void periph_init(void)
{
#      if dg_configBLACK_ORCA_MB_REV == BLACK_ORCA_MB_REV_D
#            define UART_TX_PORT    HW_GPIO_PORT_1
#            define UART_TX_PIN     HW_GPIO_PIN_3
#            define UART_RX_PORT    HW_GPIO_PORT_2
#            define UART_RX_PIN     HW_GPIO_PIN_3
#      else
#            error "Unknown value for dg_configBLACK_ORCA_MB_REV!"
#      endif


      hw_gpio_set_pin_function(UART_TX_PORT, UART_TX_PIN, HW_GPIO_MODE_OUTPUT,
                  HW_GPIO_FUNC_UART_TX);
      hw_gpio_set_pin_function(UART_RX_PORT, UART_RX_PIN, HW_GPIO_MODE_INPUT,
                  HW_GPIO_FUNC_UART_RX);


      /* Configure pin PIN_5 as a GPIO with output functionality.*/
      hw_gpio_configure_pin(HW_GPIO_PORT_1, HW_GPIO_PIN_5, HW_GPIO_MODE_OUTPUT,
HW_GPIO_FUNC_GPIO, pin_status_flag);

}


/**
 * @brief Hardware Initialization
 */
static void prvSetupHardware( void )
{

      /* Init hardware */
      pm_system_init(periph_init);
      init_wkup();
}
```

Revision History

Revision	Date	Description
1.0	08-Dec-2017	First released version
2.0	26-Nov-2018	Updated released version

Starting a Project

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

Contacting Dialog Semiconductor

United Kingdom (Headquarters)
Dialog Semiconductor (UK) LTD
Phone: +44 1793 757700

Germany
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5769 5100

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Hong Kong
Dialog Semiconductor Hong Kong
Phone: +852 2607 4271

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8200

China (Shenzhen)
Dialog Semiconductor China
Phone: +86 755 2981 3669

China (Shanghai)
Dialog Semiconductor China
Phone: +86 21 5424 9058