Tutorial

External Interruption

For the DA1468x Devices

Abstract

This tutorial should be used as a reference guide to gain a deeper understanding of the 'External Interruption' concept. As such, it covers a complete demonstrating example related to external interrupts on the DA1468x family of devices.

Figures

Figure 1: Paste Project 4
Figure 2: Compile Project 4
Figure 3: K1 Button Schematic 5
Figure 4: Terminal Interface 8
Figure 5: Multiple Interrupt Sources 10

1 Terms and Definitions

bps	Bits per Second
GPIO	General Purpose Input Output
OS	Operating System
RTOS	Real Time Operating System
SDK	Software Development Kit
UART	Universal Asynchronous Receiver - Transmitter

2 References

[1] UM-B-060, DA1468x/DA1510x PRO-Development kit, User manual, Dialog Semiconductor.

Introduction

1.1 Before You Start

Before you start you need to:

- Install the latest SmartSnippets Studio
- Download the latest SDK (currently version 1.0.12.1075)

These can be downloaded from the Dialog Semiconductor support portal.

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

The key goals of this tutorial are to:

- Provide a basic understanding of the external interruption mechanism
- Set up a working demonstration handling an external interruption and taking action
- Trigger an OS task execution based on the collected external interruption

1.2 External Interruption

An external interruption is where the processor activity is interrupted and the Cortex M0 handler mode is entered (see ARM® v6-M Architecture Reference Manual, Section B1.3.1) based on the rising and/or falling edge of a DA1468x pin. Any GPIO in the DA1468x can be used to generate an interrupt which will trigger the Cortex M0. The process is handled by the block called Wake-Up Timer (see DA1468x Datasheet, Section 19). On top of getting the processor to enter handler mode, the Wake-Up Timer will wake-up the DA1468x if it is currently in Sleep mode.

1.3 Working Demonstration

As the Bluetooth low energy framework runs under the supervision of a Real Time Operating System (RTOS), the example in this tutorial demonstrates how to trigger a task execution based on the collected event.

Clone FreeRTOS retarget Project

The first step is to create a new project. As the goal is to collect the interruption and to trigger a task handled by the RTOS, the base project will be **freertos_retarget**. This sample application starts the RTOS scheduler, performs the system initialization, and launches a task which will send a character (#) on the UART every second.

1. Start SmartSnippets Studio.

2. Import the **freertos_retarget** project. Check our getting started tutorial if you're not familiar with the procedure.

3. Copy and paste your project and call it **external_interrupt**. When pasting, make sure you paste in the same folder as the source project. In this case, SDKROOT is "C:\dev\DA1468x_DA15xxx_SDK_1.0.10.1072" and the destination folder is set to "C:\dev\DA1468x_DA15xxx_SDK_1.0.10.1072\projects\dk_apps\templates\external_interrupt".

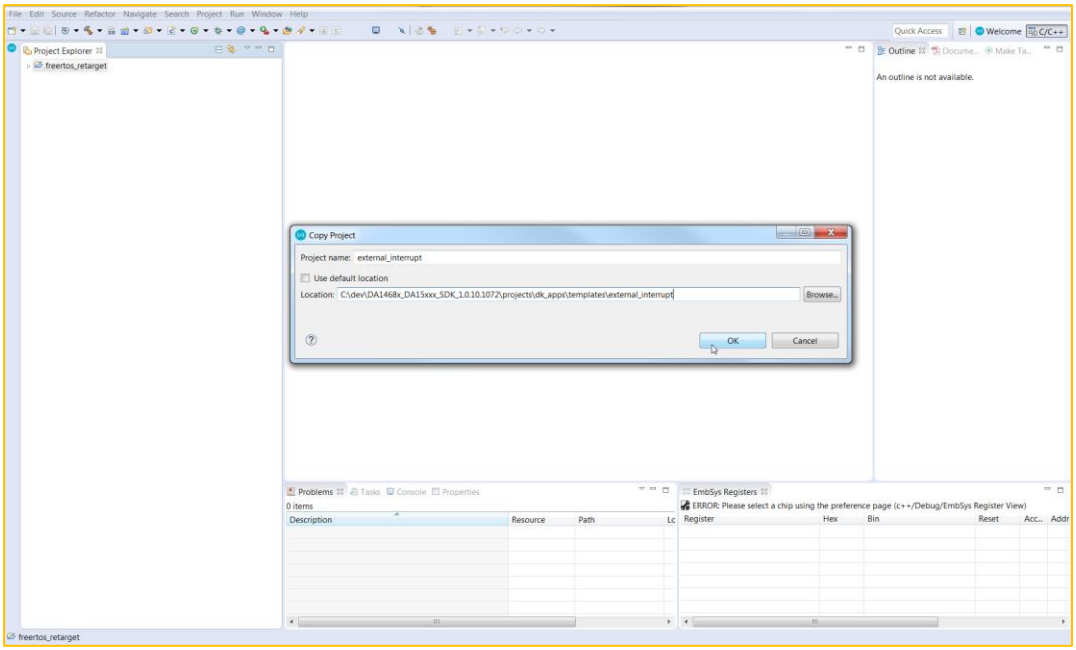

Figure 1: Paste Project

4. Compile the project for your target to check the project is set up correctly.

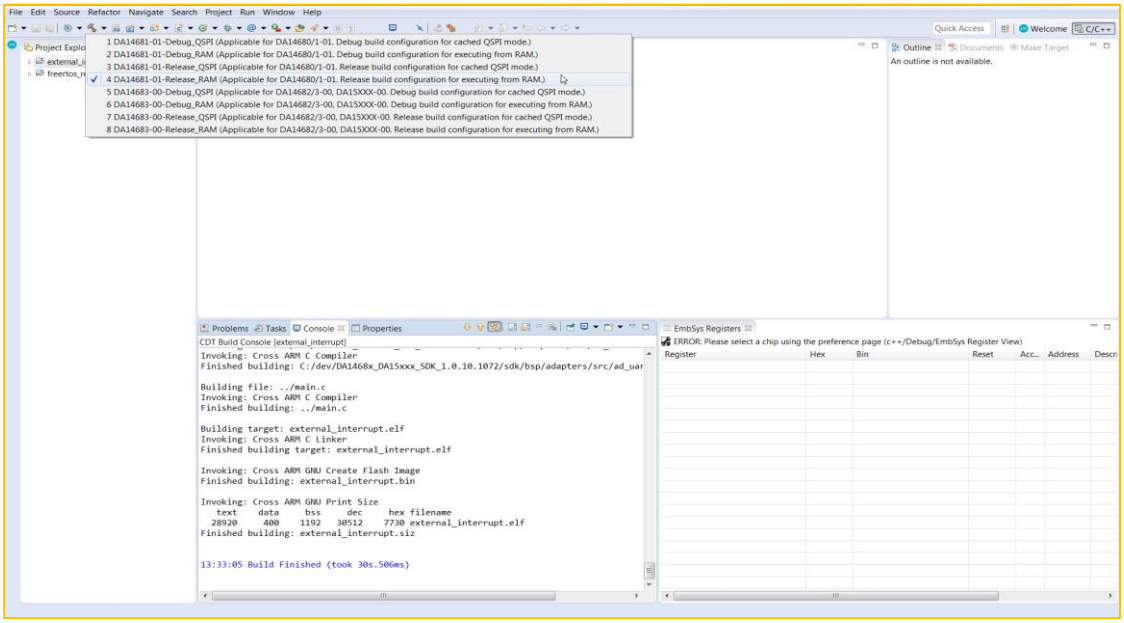

Figure 2: Compile Project

Implementation

Using the project created in the Clone FreeRTOS retarget Project section, this section demonstrates how to catch the external event and use it in the RTOS.

3.1 Existing Application

The **freertos_retarget** project demonstrates the scheduling of a timer-based task by sending a character on the UART every second. This behavior is implemented in main.c.

```
static void prvTemplateTask( void *pvParameters )
{
	OS_TICK_TIME xNextWakeTime;
	static uint32_t test_counter=0;

	/* Initialize xNextWakeTime - this only needs to be done once. */
	xNextWakeTime = OS_GET_TICK_COUNT();

  for( ;; ) {
		/* Place this task in the blocked state until it is time to run again.
		  The block time is specified in ticks, the constant used converts ticks
		  to ms.  While in the Blocked state, this task will not consume any CPU
		  time. */
		vTaskDelayUntil( &xNextWakeTime, mainCOUNTER_FREQUENCY_MS );
		test_counter++;

		if (test_counter % (1000 / OS_TICKS_2_MS(mainCOUNTER_FREQUENCY_MS))
== 0) {
			printf("#");
			fflush(stdout);
		}
	}
}
```

3.2 Configure the GPIO Generating Interruption

On the DA1468x development kits, the button **K1** is mapped to the GPIO P1_6. In order to detect an interrupt, it is necessary to configure the GPIO as an input. As the board schematic shows, the button connects the pin to the ground. We will set a pull-up on the pin to be able to detect a falling edge when the button is pressed.

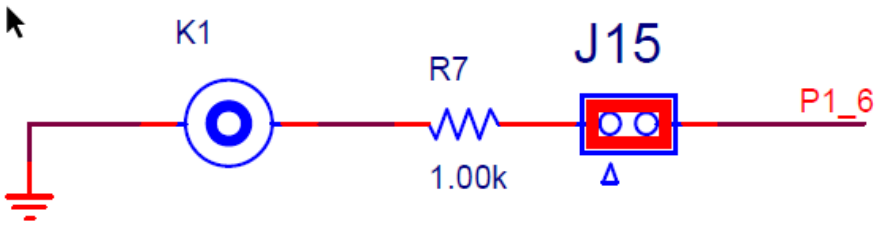

Figure 3: K1 Button Schematic

The configuration should look like:

```
/* Configure button interrupts */
hw_gpio_configure_pin(HW_GPIO_PORT_1, HW_GPIO_PIN_6,
HW_GPIO_MODE_INPUT_PULLUP, HW_GPIO_FUNC_GPIO, 1);
```

3.3 Set up the Wake-Up Timer

The next step is to initialize the Wake-Up Timer block and configure it to generate an interuption on the falling edge. Then we need to register a callback to handle the interruption, the function will be button_interrupt_cb().

```
#include "hw_wkup.h"

/* Initialize the Wake-up Timer */
hw_wkup_init(NULL);

/* Configure the Wake-up Timer to interrupt on the falling edge of P1_6 */
hw_wkup_configure_pin(HW_GPIO_PORT_1,HW_GPIO_PIN_6, 1,
HW_WKUP_PIN_STATE_LOW);

/* Register callback to handle the event from the GPIO */
hw_wkup_register_interrupt(button_interrupt_cb, 1);
```

The NULL parameter in hw_wkup_init() resets the wake-up block to its initial state. hw_wkup_configure_pin() configures the GPIO P1_6 to generate the interruption.

3.4 Handle the Interruption

button_interrupt_cb() will be executed in handling mode. In order to defer the handling of the interruption inside the RTOS, we will generate a task notification. The callback also needs to clear the current interruption.

```
void button_interrupt_cb(void)
{
      hw_wkup_reset_interrupt();
      OS_TASK_NOTIFY_FROM_ISR(task_h, 0x1, OS_NOTIFY_SET_BITS);
}
```

Note: The task_h handler is declared in the system_init() function and needs to be made global to allow button_interrupt_cb() to use it.

3.5 Using the Notification

The notification can be used by the prvTemplateTask(). We can modify the loop inside the task so that instead of sending a character on regular basis, we wait for the button push notification. Effectively the character will be sent everytime the button is pushed.


```
for( ;; ) {

        /* Wait for the external interruption notification */
        OS_TASK_NOTIFY_WAIT(0x0, OS_TASK_NOTIFY_ALL_BITS, &ulNotifiedValue,
OS_TASK_NOTIFY_FOREVER);

        /* Check the notification is the expected value */
        if(ulNotifiedValue & 0x1){
              /* Send the character on the UART */
             printf("#");
             fflush(stdout);
        }
   }
```

Eventually the code should look like this:

```
#include "hw_wkup.h"

void button_interrupt_cb(void)
{
      hw_wkup_reset_interrupt();
      OS_TASK_NOTIFY(task_h, 0x1, OS_NOTIFY_SET_BITS);
}


/**
 * @brief Template task increases a counter every mainCOUNTER_FREQUENCY_MS ms
 */
static void prvTemplateTask( void *pvParameters )
{
      OS_TICK_TIME xNextWakeTime;
      static uint32_t test_counter=0;
      uint32_t ulNotifiedValue;

      /* Configure button interrupts */
      hw_gpio_configure_pin(HW_GPIO_PORT_1,HW_GPIO_PIN_6,
HW_GPIO_MODE_INPUT_PULLUP,HW_GPIO_FUNC_GPIO, 1);

      /* Initialize and configure the Wake-up Timer */
      hw_wkup_init(NULL);
      hw_wkup_configure_pin(HW_GPIO_PORT_1,HW_GPIO_PIN_6, 1,
HW_WKUP_PIN_STATE_LOW);

      hw_wkup_register_interrupt(button_interrupt_cb, 1);

   for( ;; ) {

            /* Wait for the external interruption notification */
            OS_TASK_NOTIFY_WAIT(0x0, OS_TASK_NOTIFY_ALL_BITS,
&ulNotifiedValue,OS_TASK_NOTIFY_FOREVER);

            /* Check the notification is the expected value */
```



```
            if(ulNotifiedValue & 0x1){
                /* Send the charaacter on the UART */
                printf("#");
                fflush(stdout);
            }
        }
    }
```

3.6 Testing

Load the code on the the target development kit and start execution. Open a serial terminal of your choice with the following parameters:

- 115200 bps
- 8 bits
- no flow control
- 1 stop bit

After each press you should see an additional # character appear in your serial console as shown in Figure 4.

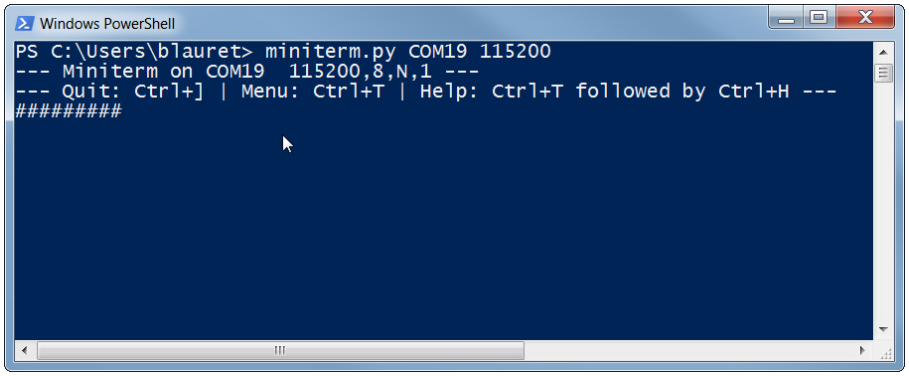

Figure 4: Terminal Interface

Handling Multiple Interruption Sources

4.1 Latching the Interrupt Source

If your application is required to handle multiple event sources, the handler will need to be aware of the source of the interruption. This can be done by latching the interrupt source. To enable this feature, it has to be defined in either custom_config_qspi.h or custom_config_ram.h.

```
#define dg_configLATCH_WKUP_SOURCE        (1)
```

4.2 Configuring Additional Sources

For this example, we will configure the GPIO P1_7 as a source of event:


```
/* Configure button interrupts 2 */
hw_gpio_configure_pin(HW_GPIO_PORT_1, HW_GPIO_PIN_7,
HW_GPIO_MODE_INPUT_PULLUP, HW_GPIO_FUNC_GPIO, 1);
```

Then we configure the Wake-up Timer block:

```
hw_wkup_configure_pin(HW_GPIO_PORT_1, HW_GPIO_PIN_7, 1,
HW_WKUP_PIN_STATE_LOW);
```

4.3 Handling the Interruption

When an interruption occurs, it is now necessary to check the source of the interruption. This is done by calling hw_wkup_get_status(). It is also necessary, since the input is latched, to clear the source pin of the interrupt. The interruption handler registered will become:

```
void button_interrupt_cb(void)
{
      uint8_t status;

      /* Request the status */
      status = hw_wkup_get_status(HW_GPIO_PORT_1);

      /* Check the status of the source 1 */
      if (status & (1 << HW_GPIO_PIN_6)) {
            /* Notify the main task */
            OS_TASK_NOTIFY_FROM_ISR(task_h, 0x1, OS_NOTIFY_SET_BITS);
            /* Clear the interrupt */
            hw_wkup_clear_status(HW_GPIO_PORT_1, (1 << HW_GPIO_PIN_6));
      /* Check the status of the source 2 */
      } else if (status & (1 << HW_GPIO_PIN_7)) {
            /* Notify the main task */
            OS_TASK_NOTIFY_FROM_ISR(task_h, 0x2, OS_NOTIFY_SET_BITS);
            /* Clear the interrupt */
            hw_wkup_clear_status(HW_GPIO_PORT_1, (1 << HW_GPIO_PIN_7));
      }
}
```

Now the application can verify the different notification and act accordingly:

```
/* Check the notification is button 1 */
 if(ulNotifiedValue & 0x1){
       /* Send the character on the UART */
      printf("#");
      fflush(stdout);
/* Check the notification is button 2 */
} else if(ulNotifiedValue & 0x2) {
      /* Send the character on the UART */
     printf("$");
```



```
        fflush(stdout);

}
```

This example results in different characters being sent depending on the button pressed, as depicted in multiple_interrupt

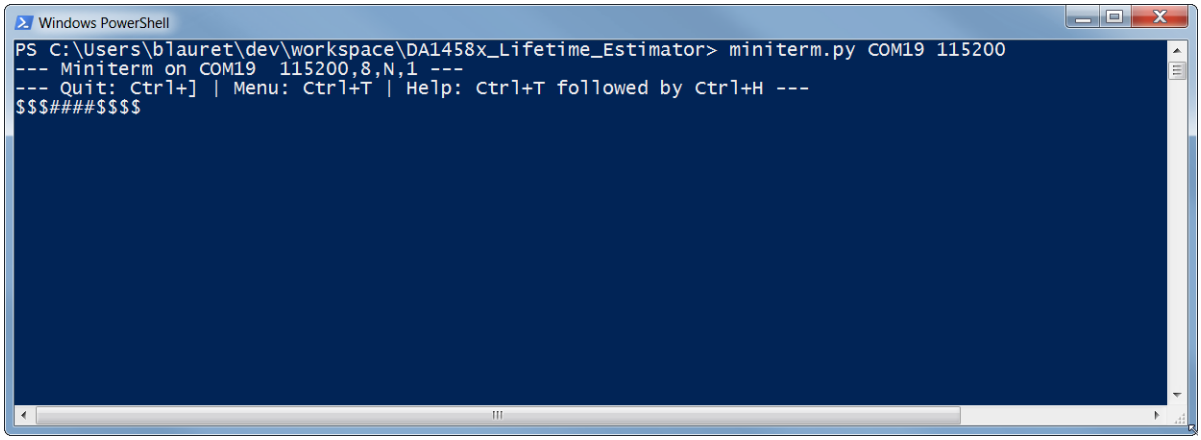

Figure 5: Multiple Interrupt Sources

Revision History

Revision	Date	Description
1.0	5-January-2018	First released version

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