MSP430X interruption/LPMx

Antoine Bernabeu

October 22, 2024

1 Contexte

MSP430 microcontrollers are designed for ‘ultra-low energy’ applications. In this context, it is common for the application to minimise its energy consumption, particularly through the use of LPMx (low power mode).

2 Problem

The MSP430 uses interrupt vectors which group together the different possible sources for an interrupt. With Trampoline, when you want to use an interrupt to wake up the CPU, you use a Boolean on the interrupt vector. So, whatever the source of the interrupt, if the boolean is set to TRUE, the CPU will be woken up.

For example, I want to wake up the CPU by pressing push button S1. If I press push button S2, the CPU wakes up. A first approach might be to mask the interrupts from push button S2. However, let’s imagine another case where we also want to use push button S2.

It is then impossible to wake up the CPU simply by pressing push button S1.

With the code in Figure 1, if the push button S2 is pressed, the green LED will light up, meaning that the CPU woke up from LPMx which is not intented.

3 Proposal

To differentiate the source of the interrupt in order to wake the CPU according to it, we add to each interrupt a boolean named *EXIT_FROM_LPM* which is by default **FALSE**. When setting this boolean to **TRUE**, the interrupt will wake up the CPU when finishing its execution. OIL code in Figure 2 illustrates how to set up 2 interruptions, from the same vector, one that wake up the CPU and the other that doesn’t wake up the CPU.

Figure 1: Code exemple

```
#define APP_Task_blink_START_SEC_CODE
#include "tpl_memmap.h"
TASK(blink)
{
  while(1){
        __bis_SR_register(LPM3_bits + GIE);
        P1OUT ^= 1 << 1;      /* toggle green led */
  }
        TerminateTask();
}
#define APP_Task_blink_STOP_SEC_CODE
#include "tpl_memmap.h"

#define APP_ISR_buttonS1_START_SEC_CODE
#include "tpl_memmap.h"
ISR(buttonS1)
{
        P1OUT ^= 1; //toggle red led
}
#define APP_ISR_buttonS1_STOP_SEC_CODE
#include "tpl_memmap.h"

#define APP_ISR_buttonS2_START_SEC_CODE
#include "tpl_memmap.h"
ISR(buttonS2)
{
        /* Some stuff to do */
}
#define APP_ISR_buttonS2_STOP_SEC_CODE
#include "tpl_memmap.h"
```

Figure 2: OIL ISR description with boolean for LPM

```
ISR buttonS1 {
  CATEGORY = 1;
  PRIORITY = 1;
  SOURCE = PORT5_VECTOR {
    BIT = 6;
  };
  EXIT_FROM_LPM = TRUE;
};

ISR buttonS2 {
  CATEGORY = 1;
  PRIORITY = 1;
  SOURCE = PORT5_VECTOR {
    BIT = 65
  };
  EXIT_FROM_LPM = FALSE;
};
```

3.1 Change for ISR category 1

For category 1 ISR, the change is straightforward. We modified the *direct_irq_handler* template such that after execution the interruption code, if the boolean *EXIT_FROM_LPM* is set to **TRUE**, we change the special register (SR) and we clear bit 7:4.

And exemple of the irq handler generated is illustrate in code from Figure 3. We note the change with the *bicx* instruction that change the SR register within the process stack which is 5 registers deep. If using the large model of msp430x, that mean the SR register is 20 bytes deep within the stack.

3.2 Change for ISR category 2

For ISR category 2, this is more complicated as they are considered as tasks and scheduled according to their priority. We then chose to modify the *system call handler*. We target only the *CallterminateISR2* system call. When we enter the *system call handler*, we push the *system call id* to the kernel stack to save it for later. If the *system call id* is the same as *CallTerminateISR2*, we check either we need to wake up before running *tpl_run_elected*, ie. before we are running the next task at the end of the ISR2. We get back the *system call id* from the kernel stack, compare it to the *CallTerminateISR2* id. If it match, we call a function that return either 1 or 0 if we need to wake up (1 we need to wake up, 0 we don't). The result is pushed to the kernel stack. We pop this result after running *tpl_run_elected*, and if it is 1, we modify the SR register of the context of the next running task. The SR register is 12 registers deep, thus 48 bytes if using large model.

This manipulation need and extra variable in the descriptor of ISR2 and tasks. For

Figure 3: Exemple of handler for 2 interruptions from the same vector (here PORT5). If the source is bit 5, we want to wake up from LPM (EXIT FROM LPM is set to TRUE). If the source is bit 6, we don't wake up the CPU after the interruption.

```
/*============================================================================
 * IRQ Handler for ISR category 1:
 * buttonS2, bit 5
 * buttonS1, bit 6
 * with source vector PORT5_VECTOR
 ****************************************************************************/

.extern buttonS2_function
.extern buttonS1_function

.section .irq_func, "ax"
.global tpl_direct_irq_handler_PORT5_VECTOR
.type   tpl_direct_irq_handler_PORT5_VECTOR, %function
/*----------------------------------------------------------------------------
 */
tpl_direct_irq_handler_PORT5_VECTOR:

/*----------------------------------------------------------------------------
 * -1- Push volatile registers
 */
  pushm.a   #5, r15
/*----------------------------------------------------------------------------
 * -2- Get the highest priority interrupt of the port from P5IV
 * and jump to the handler
 */
  addx.a       &__P5IV, pc
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR       /* bit 0 */
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR       /* bit 1 */
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR       /* bit 2 */
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR       /* bit 3 */
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR       /* bit 4 */
  jmp       tpl_p5_5_handler                               /* bit 5 */
  jmp       tpl_p5_6_handler                               /* bit 6 */
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR       /* bit 7 */
tpl_p5_5_handler:
  calla      #buttonS2_function
/*----------------------------------------------------------------------------
 * -2b- exit from LPM if required
   we pushed 5 volatile registers so SR is 2*5 bytes deep
 */
  bicx      #0xF0, 20(SP)
  jmp       tpl_direct_irq_handler_exit_PORT5_VECTOR
tpl_p5_6_handler:
  calla      #buttonS1_function
/*----------------------------------------------------------------------------
 * -2b- exit from LPM if required
   we pushed 5 volatile registers so SR is 2*5 bytes deep
 */
/*----------------------------------------------------------------------------
 * -3- Pop volatile registers
 */
tpl_direct_irq_handler_exit_PORT5_VECTOR:
  popm.a    #5, r15
/*----------------------------------------------------------------------------
 * -4- Return.
 */
  reti
```

tasks, this variable is useless. For ISR2, it is set to 0 or 1 in the IRQ handler accordingly with the value of the boolean *EXIT_FROM_LPM* depending on the interruption source.