Operating systems for embedded systems

Danilo Moceri (ID: 242550)

December 4, 2016

Assignment #1

An alarm system has to be developed to monitor the voltage of a given analog input, and to inform the user using visual indications through RGB led.

Introduction

To solve the assignment I have developed two different solutions below explained. To make easier tests two solutions can be selected simply changing a define in the file *"app.h"* as shown:

```
#include        "fsl_interrupt_manager.h"

/*      Uncomment this define to compile the second solution    */
//#define       TIMER_INTERRUPT

#define         FREQ_10HZ       50
#define         FREQ_20HZ       25
#define         FREQ_0HZ        0
```

Both developed solutions use an ADC peripheral with interrupt mode enabled. To manage the ADC module I have used a Task called *adcTask*. The task code will be compile in different mode depending of the TIMER_INTERRUPT define. It's important to observe that *adcTask* is an infinite loop task, its principal target is to enable continuously the ADC module, after this operation the task calls System Call *OSSemPend* and it waits until that ADC conversion will be completed.
In fact while the ADC module is working it's possible to release the CPU that can handle other tasks.

```
static void adcTask(void *p_arg){

        //other code........
        while(DEF_TRUE){
                /*      Enable the ADC module    */
                adcStart();
                /*      Pend the semaphore to release the CPU   */
                OSSemPend(&sem, 0, OS_OPT_PEND_BLOCKING, 0,&err);
                //other code........
        }
}
```

The interrupt that comes from ADC module is managed by the callback function called *adcInterruptHandler*, its code is shown below and it's possible to see the System Call *OSSemPost* that allows the semaphore unlocking. Then System Call *OSIntExit* determines the highest priority task in state ready to run and executes it.

```
void adcInterruptHandler(void){

        //other code........
        /*      read the ADC value      */
        adcValue = ADC0_RA;
        /*      post the semaphore for adc task         */
        OSSemPost(&sem, OS_OPT_POST_1 | OS_OPT_POST_NO_SCHED, &err);
        /*      renable interrupts   */
        CPU_CRITICAL_EXIT();
        /*      notify to scheduler the end of an ISR    */
        OSIntExit();
}
```

1 Solution

The first developed solution uses another task, called *ledControlTask*, to manage the LEDs blinking.
To synchronize the two tasks I have used a semaphore that allows to decide which task has to start before, in this way is possible to avoid that *ledControlTask* starts to blink LEDs when the level voltage on PTB2 is still unknown.
To obtain the right delay I used the System Call *OSTimeDlyHMSM* for LEDs blinking, as it is shown below:

```
static void ledControlTask(void *p_arg){
        //other code........

        /*      Pend the semaphore startSem and wait until that it will be unlocked from the
            ADC task */
        OSSemPend(&startSem, 0u, OS_OPT_PEND_BLOCKING, 0u,&err);

        while(DEF_TRUE){
                /* If some led has to blink      */
                if(currentFreq != FREQ_0HZ){
                        GPIO_DRV_TogglePinOutput(currentLed);
                        OSTimeDlyHMSM(0u, 0u, 0u, currentFreq, OS_OPT_TIME_HMSM_STRICT, &err);
                }
        }
}
```

2 Solution

The second developed solution uses a Periodic Interrupt Timer (PIT) to allow LEDs blinking.
The PIT is a peripheral that provides a counter which is decremented every clock cycle, it generates an output signal when it reaches the zero value. The output signal in this case is used to trigger an interrupt.

The interrupt generated by PIT module calls a callback function that is able to switch the LEDs.
To obtain this functionality it is necessary to calculate the right value to set as interrupt period. For developing this application the value is set into PIT_LDVAL1 register and it is calculated as:

$$PIT\_LDVAL1 = Clock_{frequency} \cdot Interrupt_{period} \tag{1}$$

Where $Clock_{frequency}$ is the peripherals clock frequency and it is set to $60 \cdot 10^6 Hz$, while $Interrupt_{period}$ is the interrupt period wanted.
The values calculated are set into the PIT_LDVAL1 register depending of the voltage read by the ADC module, as it is possible to see below:

```
static void adcTask(void *p_arg){
        //other code........
        #ifdef TIMER_INTERRUPT
                //other code........
                /*      This function is used to calculate which
                led has to blink and its blink rate depending of the voltage read*/
                computeLedPeriod();
                /*      Switch the current frequency to set the right period*/
                switch(currentFreq){

                case FREQ_10HZ:
                        /*      If the frequency is 10Hz the interrupt has to occur
                        * every 50ms so the register has to be set to 3.000.000 */
                        PIT_LDVAL1 = 0x2DC6C0;
                        /*      Turn on PIT module      */
                        PIT_TCTRL1 |= 0x01;
                        break;
                case FREQ_20HZ:
                        //other code........
```

To manage the interrupt that comes from the PIT module I have written an interrupt handler, it is called *timerInterruptHandler* and its principal goals are toggle the right LED and reset the interrupt flag from PIT_TFLG1 register.
The *timerInterruptHandler* code is shown below:

```
void timerInterruptHandler(void){
        //other code........
        /*      If the interrupt comes from PIT 1 */
        if(PIT_TFLG1){
                /*      Toggle the led */
                GPIO_DRV_TogglePinOutput(currentLed);
                /*      Reset the interrupt flag*/
                PIT_TFLG1 = 0x01;
        }
        //other code........
}
```

With the second solution shown is possible to avoid the use of another task, in this way surely is possible to reduce the number of resources used and the operative system is less used than the first solution.