EPOS Lab: Light Keyboard on EPOSMote III

This exercise aims at making clear the differences between two typical architectures for cyclic-executive-based embedded systems: round-robin and round-robin with interrupts. The basic idea is to associate leds and keys in such a way that a led is on when the corresponding key is pressed (for some time).

Architecture I: Round-Robin

This first version uses no interrupts. It simply polls the state of buttons in a tight loop, identifying those that are pressed and turning on the corresponding leds.

Pseudo code

```
void main(void) {
    while(1)
        leds_on(buttons_chk());
}
```

The buttons_chk function reads the data register from the I/O port to which the buttons are connected, returning a mask of pressed buttons, while leds_on takes the activation mask to set the corresponding bits of the data register from the I/O port to which the leds are connected.

Solutions:

C:

```
#define  GPIO_AFSEL (*(volatile unsigned int *)(0x400db420))
#define  GPIO_IC (*(volatile unsigned int *)(0x400db41c))
#define  GPIO_PORTC (*(volatile unsigned int *)(0x400db000))
#define  GPIO_DIR (*(volatile unsigned int *)(0x400db400))
#define  GPIO_F (*(volatile unsigned int *)(0x400db0cc))
#define  GPIO_IRQ_DETECT_ACK (*(volatile unsigned int *)(0x400db718))

enum {
    TURN_ON      = 0xff,
    LED_PIN      = 3,
    BTN_PIN      = 4,
    LED_PIN_MASK = 1 << LED_PIN,
    BTN_PIN_MASK = 1 << BTN_PIN,
};

void config() {
    GPIO_AFSEL &= ~(LED_PIN_MASK | BTN_PIN_MASK);
    GPIO_DIR |= LED_PIN_MASK;
    GPIO_IC = LED_PIN_MASK ;
    GPIO_IRQ_DETECT_ACK &= ~(LED_PIN_MASK << 16);
}

void set_led(bool on) {
    volatile unsigned int *aux = (volatile unsigned int *) (0x400db000 + (LED_PIN_MASK <<
2));
    if (on)
        *aux |= TURN_ON;
    else
        *aux &= ~TURN_ON;
```

```
}

void blink(){
    set_led(true);
    for(int i=0;i<999999;i++);
    set_led(false);
    for(int i=0;i<999999;i++);
}

void read_btn(){
    volatile unsigned int *aux = (volatile unsigned int *) (0x400db000 + (BTN_PIN_MASK <<
2));
    if(!(*aux))
        blink();
}

int main() {
    config();
    while(1) {
        read_btn();
    }
    return 0;
}
```

C++

Architecture II: Round-Robin with Interrupts

To make things more interesting, we will now wait for a button to be pressed for at least 4 seconds before turning on the corresponding led. This is more easily achievable with another software architecture: round-robin with interrupts.

Interrupt Handling

First of all, you need to know how to handle external interrupts. Connect one of the STK500 buttons to one of the eMote's external interrupt pins and write a simple routine to handle that IRQ. You must configure this external interrupt in order that a button release (a rising edge on the interrupt pin) triggers an event (e.g. prints a bit pattern in the STK500's leds).

```
void main(void) {
    int_handler(BUTTON, &button_handler);
    int_enable();
    while(1);
}

void button_handler(void) __ISR__ {
    leds_on(buttons_chk());
}
```

Timer Operation

After handling interrupts, you will need to get acquainted with basic timer operations. Set up a timer event in the eMote so that the leds print out a value after holding a button for 4 seconds.

Timer Frequency

A timer counts up a value every two clock cycles, and can be set-up to count from an arbitrary value (Time) up to it's maximum value (256 in an 8-bit timer). The timer can be configure to trigger an interrupt when it overflows or when its counter value matches a defined value. The timer input clock can be prescaled by N from the System Clock, meaning that N system cycles correspond to one timer clock cycle. Thus, considering the timer is configured to issue interrupts on overflows, the formula for the interrupt frequency in an 8-bit timer is Clk/N*256.

Some tips:

Find out the registers for the timer counter, interrupt generation control, and clock source (the CC2538 has a lot of independent timers, you may want to use the 8-bit Timer 0)
Set up the timer to issue interrupts on counter overflow
Set up a interrupt handler for a button. It should start/stop the timer when the button is pressed/released. This can be achieved by switching the interrupt triggering condition inside the button interrupt handler (see the External Interrupt Control Register description in the datasheet)
Set up a interrupt handler for the timer. After X interrupts, it should turn on the leds. The value of X can be calculated using the result of the formula above.

Solutions

C
C++