EPOS Lab: Led Blinking on EPOSMote III

This is a very simple exercise whose purpose is to get you acquitted with the process of compiling, loading and checking a program for embedded systems. The task is to solely set up a led to blink indefinitely. But before starting, make sure that you have gone through EPOSMoteIII Quick-Start for basics on how to program it.

Pseudo code

Let's begin with a pseudo code (actual coding is up to you):

▯▯▯▯▯▯▯

```
int main() { while(1) { led_on(); delay(); led_off(); delay(); } return 0; }
```

The pseudo-code is self-explanatory: an infinite loop alternating commands to turn on the led, cause some delay, turning off the led, and cause some delay. The really interesting things are the functions led_on() and led_off().

EPOSMote III is built around the TI CC2538 SoC and has a red LED connected to pin **3** of GPIO port **C** . Therefore, you can check this manual for instructions on how to access that GPIO.

For a while, until we get to EPOS Lab: Timers, you can use an empty, non-optimized loop to implement `delay()`.

You can also use EPOS' ((EPOSMoteIII Quick-Start|Hello World) app to get an easy framework to program your solution but refrain from using EPOS' GPIO and timing services for a while. Your goal here is to get acquited with the manual and the tool-chain.

Compiling and Uploading

Follow the steps in the EPOSMoteIII Quick-Start to compile your application, and this build a tailored instance of EPOS, and to upload it to the mote. Check the console for details about the tools used to prepare the image and to upload it to the mote.

Schematic

Not needed for this lab (EPOSMote III has a built-in LED).

Part List

Not needed for this lab (EPOSMote III has a built-in LED).

Assembly

Not needed for this lab (EPOSMote III has a built-in LED).

Solutions

Assembly:

▯▯▯▯▯▯▯

```
int main() { ASM("\t\n\ ldr r0, =0x400DB420 // load GPIO_AFSEL reg controls selects hardware control for specific GPIO\t\n\ ldr r1, [r0] \t\n\ mov r2, #3 // put #3 in reg r2 \t\n\ lsl r2, r2, #3 // left shift reg r2 to mask pin 3 \t\n\ bic r1, r1, r2 // change bit 3 using mask \t\n\ str r1, [r0] // write to GPIO_AFSEL \t\n\ sub r0, r0, #0x20 // sub 20 from GPIO_AFSEL to GPIO_DIR controls direction of GPIO pins \t\n\ ldr r1, [r0] \t\n\ orr r1, r1, r2 // mask bit 3 with r2 \t\n\ str r1, [r0] \t\n\ ldr r0, =0x400DB41C // load GPIO_IC interrupt clear register \t\n\ mov r1, r2 // load mask into r1 \t\n\ str r1, [r0] // load r1 into GPIO_IC \t\n\
add r0, r0, #0x2fc // add to GPIO_IC to go to GPIO_IRQ_DETECT_ACK \t\n\ ldr r1, [r0] \t\n\ lsl r3, r2,
#16 // shift mask left by 16 bits \t\n\ bic r1, r1, r3 // apply mask to GPIO_IRQ_DETECT_ACK \t\n\ str r1,
[r0] \t\n\ ldr r0, =0x400DB000 // load GPIO_PORT_C \t\n\ lsl r2, r2, 2 // shift mask left by 2 \t\n\ add r0,
r0, r2 // add mask to GPIO_PORT_C \t\n\ "); while(1){ ASM("\t\n\ mov r1, 0xff // turn on led \t\n\ str r1,
[r0] \t\n\ "); for(unsigned int i=0;i<0xffffff;i++) ASM("nop"); ASM("\t\n\ mov r1, 0x00 //turn off led \t\n\
str r1, [r0] \t\n\ "); for(unsigned int i=0;i<0xffffff;i++) ASM("nop"); } return 0; }
```

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_IRQ_DETECT_ACK (*(volatile unsigned int
*)(0x400db718)) enum { TURN_ON = 0xff, PIN = 3, PIN_MASK = 1 << PIN, }; void config() {
GPIO_AFSEL &= ~PIN_MASK; GPIO_DIR |= PIN_MASK; GPIO_IC = PIN_MASK;
GPIO_IRQ_DETECT_ACK &= ~(PIN_MASK << 16); } void set_led(bool on) { volatile unsigned int *aux
= (volatile unsigned int *) (0x400db000 + (PIN_MASK << 2)); if (on) *aux |= TURN_ON; else *aux &=
~TURN_ON; } int main() { config(); while(1) { set_led(true); for(int i=0;i<999999;i++); set_led(false);
for(int i=0;i<999999;i++); } return 0; }
```