LINGI 2315 - Homework 3 - My RTOS App **P1**

Name	Delcoigne Ben	Noma	3877 1700

Description of the task and other functions that implement the control of the blinking frequency of LED[0] with relevant screenshots of your code

On the hardware part, I used the sof file we used on all other homeworks. No changes were done there (just make sure there is a pio_0 of 27 or 32 bits for the frequency, I took 32 bits)

I created a new function in charge of modifying the led frequency :

```
void Task_FPGA_Button(void)
{
    MBX_t    *PrtMbx;
    intptr_t  PtrMsg = (intptr_t) NULL;
    SEM_t    *PtrSem;

    PrtMbx = MBXopen("MyMailbox", 128);
    PtrSem = SEMopen("MySemaphore");

    for( ;; )
    {
        SEMwait(PtrSem, -1);            // -1 = Infinite blocking
        SEMreset(PtrSem);
        MTXLOCK_STDIO();
        printf("Receive IRQ from Button %d and send message (Core = %d)\n", (int) alt_read_word(fpga_buttons) - 1, CORE_getID());  // T
        //I will now change the blinking frequency:

        uint32_t button = alt_read_word(fpga_buttons);
        uint32_t frequency = alt_read_word(fpga_pio);

        if (button == 1){
            frequency  += PIO_0_FREQ/10;
            alt_write_word(fpga_pio, frequency);
        }
        if (button == 2){
            frequency -= PIO_0_FREQ/10;
            alt_write_word(fpga_pio, frequency);
        }

        MTXUNLOCK_STDIO();

        MBXput(PrtMbx, PtrMsg, -1);     // -1 = Infinite blocking
    }
}

/* ------------------------------------------------------------------------------------------ */
```

This task runs constantly, but as you can notice, it is blocked by a semaphore. This semaphore is unlocked by the button callbackinterrupt:

Concretely, fpga_button has an infinite loop. At SEM_wait, the loop blocks and is waiting for someone to tell it to go on. That is done by a whole other thread (the one that handles interrupts).Once we have a button interrupt, we SEMost, which releases the semaphore and lets the for loop make one revolution. The cycle repeats.

```
void button_CallbackInterrupt (uint32_t icciar, void *context)
{
    SEM_t    *PtrSem;

    // Clear the interruptmask of PIO core
    alt_write_word(fpga_buttons + PIOinterruptmask, 0x0);

    // Enable the interruptmask and edge register of PIO core for new interrupt
    alt_write_word(fpga_buttons + PIOinterruptmask, 0x3);
    alt_write_word(fpga_buttons + PIOedgecapture, 0x3);

    PtrSem = SEMopen("MySemaphore");
    SEMpost(PtrSem);
}
```

I also point out to the fact that the task was created, but is nowhere called, that's why I make core 1 run it by adding the following code in main_mabassi.c

```
    Task = TSKcreate("Task FPGA Button", 1, 8192, &Task_FPGA_Button, 0);
    TSKsetCore(Task, 1);                            /* Create new task, will always run on core #1  */
    TSKresume(Task);                                /* when BMP (G_OS_MP_TYPE == 4 or 5)            */

    Task = TSKcreate("Task read the sensor", 1, 8192, &Task_ReadSensor, 0);
    TSKsetCore(Task, 1);                            /* Create new task, will always run on core #1  */
    TSKresume(Task);

    Task = TSKcreate("Task display the sensor", 1, 8192, &Task_DisplaySensor, 0);
    TSKsetCore(Task, 0);                            /* Create new task, will always run on core #1  */
    TSKresume(Task);
```

(I also added the code for readsensor and displaysensor which are for the second part of the HW)

LINGI 2315 - Homework 3 - My RTOS App **P2**

Name	Delcoigne Ben	Noma	38771700

Description of the 2 tasks and other functions that implement the reading and display of 1-axis acceleration with relevant screenshots of your code.

First step was to make the drivers working to be able to interact with the gsensor. I did just like in demo 8 (you can see what I did since I possted a few steps in the forum). But basically:

Reuse the DMA init code from demo8 (which is just above in the screen but takes a lot of space in the screenshot if I add it), and init the i2c and XL345.
Note* you must add the libraries in the makefile

```
void Task_ReadSensor(){
    //************************************
    i2c_init(0, 10, 400000);
    int init = XL345init();

    MBX_t *Accelbox;
    Accelbox = MBXopen("MailboxAccel",128);

    if(init !=0){
        printf("Init of 345 didn't work");
    }
    int x,y,z;
    x=0;
    y=0;
    z=0;

    for(;;){

        int err = XL345read(&x,&y,&z);
        if(err!=0){
            //printf("I've had trouble reading the XL345");
        }
        else{
            int out[3];
            out[0]=x;
            out[1]=y;
            out[2]=z;

            MBXput(Accelbox, (intptr_t)(out),-1);

        }
        TSKsleep(OS_MS_TO_TICK(250));
    }
}
```

```
//************************READ THE SENSOR**************************
void Task_DisplaySensor(void){

    MBX_t *Accelbox;
    Accelbox = MBXopen("MailboxAccel",128);

    int *p; //creates an array

    intptr_t Message;


    for(;;){
        int err = MBXget(Accelbox, &Message, -1);

        if (err!=0){
            printf("Error recieving from mailbox");
        }
        else{
            MTXLOCK_STDIO();
            p = (int*)Message;

            uint32_t switches = alt_read_word(fpga_sw);

            switches = switches%3;



            printf("\nReading from axis %d : %d \n", switches, p[switches]);
            MTXUNLOCK_STDIO();

        }
    }
}
```

Once done, I create a mailbox which will be used to communicate with the display task. I then read what's in the sensor, and send it to the mailbox.
On the other end, I must create a task that reads the mailbox and displays the text. (on the right). Both these tasks were initialized in main_mabassi.c (see previous page). For fun, I put one task on core 1 and the other on core0. I notice that the code works and thus the communication is indeed done between multiple cores. Great!

Litte lote on task_displaysensor:
I first setup all my variables, then indefinetly (and with no time.sleep()), I open the last message from the mailbox and print it out. In order to choose the right axis, I read the fgpa_switches (which were setup just like fpga_Buttons and fpga_pio0), and %3 in order to choose one of the axis .

Note:
The MTXLOCK_STDIO() are used to lock onto the standard output: this is to avoid multiple threads trying to write text at the same time (and thus displaying characters interleaved with others).