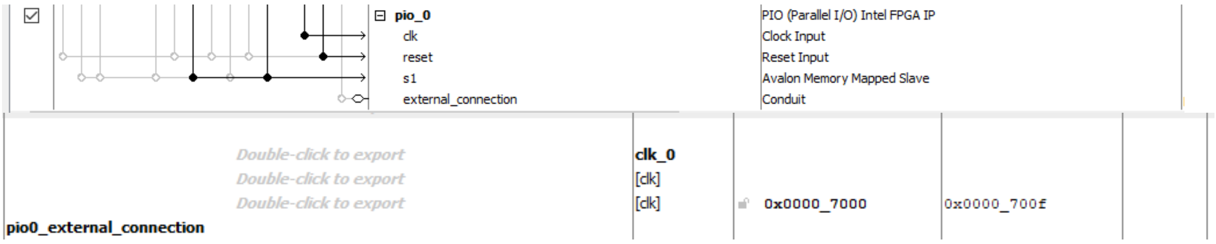
LINGI 2315 - Homework 1 - My Linux App

P1

Name	Delcoigne Ben	Noma	38771700

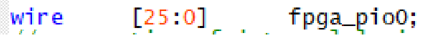
Description of the hardware modifications with relevant screenshots

In the platform designer I added a new input (26 bit PIO) which will later be used to set the led frequency. I mapped it to the first free address: 0x700

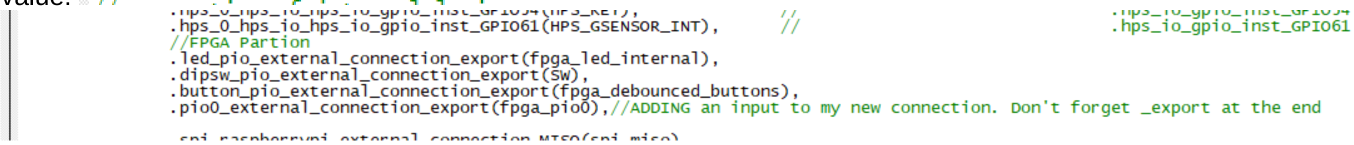


```
☑   pio_0                                   PIO (Parallel I/O) Intel FPGA IP
      clk                                   Clock Input
      reset                                 Reset Input
      s1                                    Avalon Memory Mapped Slave
      external_connection                   Conduit

                 Double-click to export     clk_0
                 Double-click to export     [clk]
                 Double-click to export     [clk]      0x0000_7000     0x0000_700f
pio0_external_connection
```

This was actually a part of the tutorial, I just set the length to 26 bits instead of 32.
In the HDL file, I added a line that “imports” that value, and I make it so that the clock reset time now depends on that value:

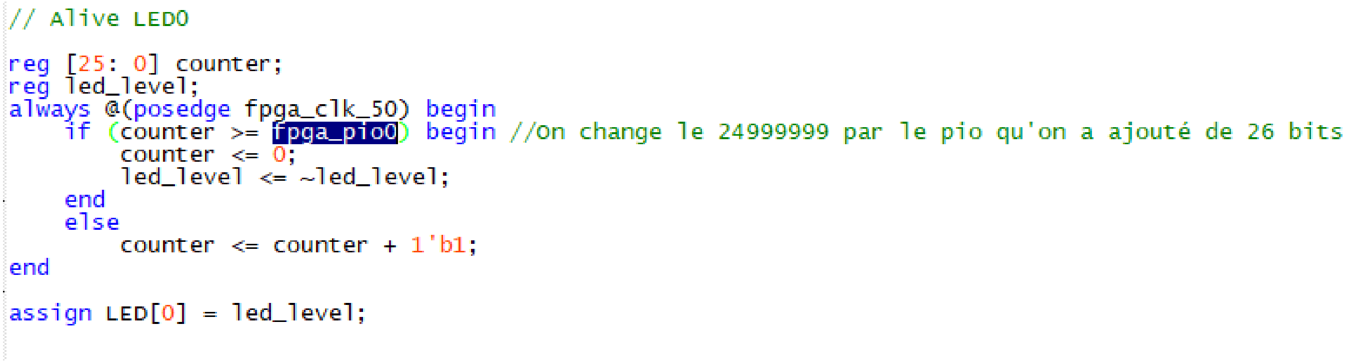


```
wire      [25:0]      fpga_pio0;
```



```
.hps_0_hps_io_hps_io_gpio_inst_GPIO61(HPS_GSENSOR_INT),      //                    .hps_io_gpio_inst_GPIO61
//FPGA Partion
.led_pio_external_connection_export(fpga_led_internal),
.dipsw_pio_external_connection_export(SW),
.button_pio_external_connection_export(fpga_debounced_buttons),
.pio0_external_connection_export(fpga_pio0),//ADDING an input to my new connection. Don't forget _export at the end
```

Make the LED reset frequency depend on that value instead of just a hardcoded 2499999 value.



```
// Alive LED0

reg [25: 0] counter;
reg led_level;
always @(posedge fpga_clk_50) begin
    if (counter >= fpga_pio0) begin //On change le 24999999 par le pio qu'on a ajouté de 26 bits
        counter <= 0;
        led_level <= ~led_level;
    end
    else
        counter <= counter + 1'b1;
end

assign LED[0] = led_level;
```

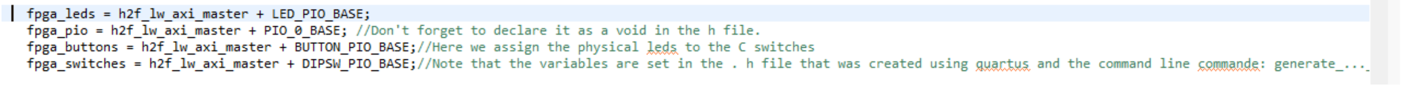
LINGI 2315 - Homework 1 - My Linux App

P2

Name	Delcoigne Ben	Noma	38771700

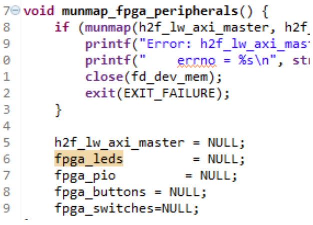
Description of the software modifications with relevant screenshots of your code

In the software, I first had to link the physical hardware IO's to variables in the code. This is done with the following snippet: (part of the mmap_fpga_peripherals()) function.



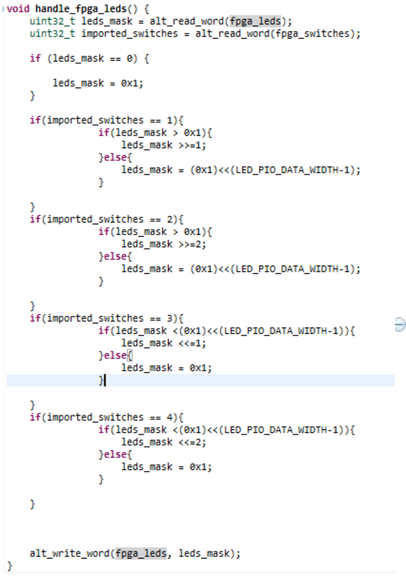
```
fpga_leds = h2f_lw_axi_master + LED_PIO_BASE;
fpga_pio = h2f_lw_axi_master + PIO_0_BASE; //Don't forget to declare it as a void in the h file.
fpga_buttons = h2f_lw_axi_master + BUTTON_PIO_BASE;//Here we assign the physical leds to the C switches
fpga_switches = h2f_lw_axi_master + DIPSW_PIO_BASE;//Note that the variables are set in the . h file that was created using quartus and the command line commande: generate_..._
```

Note: I had to initialize the variables as void pointers in the .h file
I also set these new variables to NULL in the unmap part of the code.

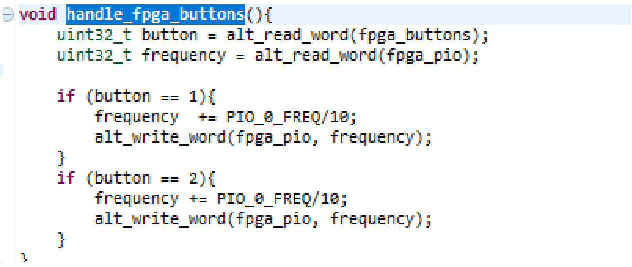


```
void munmap_fpga_peripherals() {
    if (munmap(h2f_lw_axi_master, h2f
        printf("Error: h2f_lw_axi_mas
        printf("    errno = %s\n", st
        close(fd_dev_mem);
        exit(EXIT_FAILURE);
    }

    h2f_lw_axi_master = NULL;
    fpga_leds          = NULL;
    fpga_pio          = NULL;
    fpga_buttons = NULL;
    fpga_switches=NULL;
```

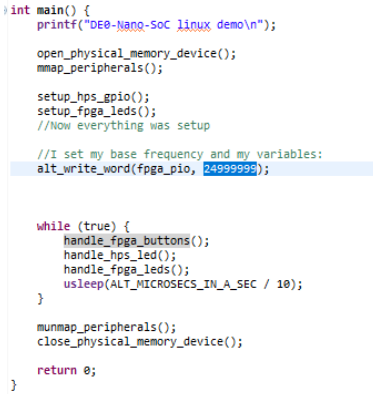
The classical LED code was changed to depend on the imported switches (used as a binary counter)
I also created a function that handles the buttons and changes the frequency of fpga_pio_0 (used in the Verilog code)



```
void handle_fpga_leds() {
    uint32_t leds_mask = alt_read_word(fpga_leds);
    uint32_t imported_switches = alt_read_word(fpga_switches);

    if (leds_mask == 0) {

        leds_mask = 0x1;
    }

    if(imported_switches == 1){
                if(leds_mask > 0x1){
                    leds_mask >>=1;
                }else{
                    leds_mask = (0x1)<<(LED_PIO_DATA_WIDTH-1);
                }

    }
    if(imported_switches == 2){
                if(leds_mask > 0x1){
                    leds_mask >>=2;
                }else{
                    leds_mask = (0x1)<<(LED_PIO_DATA_WIDTH-1);
                }

    }
    if(imported_switches == 3){
                if(leds_mask <(0x1)<<(LED_PIO_DATA_WIDTH-1)){
                    leds_mask <<=1;
                }else{
                    leds_mask = 0x1;
                }

    }
    if(imported_switches == 4){
                if(leds_mask <(0x1)<<(LED_PIO_DATA_WIDTH-1)){
                    leds_mask <<=2;
                }else{
                    leds_mask = 0x1;
                }

    }



    alt_write_word(fpga_leds, leds_mask);
}
```



```
void handle_fpga_buttons(){
    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);
    }
}
```



```
int main() {
    printf("DE0-Nano-SoC linux demo\n");

    open_physical_memory_device();
    mmap_peripherals();

    setup_hps_gpio();
    setup_fpga_leds();
    //Now everything was setup

    //I set my base frequency and my variables:
    alt_write_word(fpga_pio, 24999999);



    while (true) {
        handle_fpga_buttons();
        handle_hps_led();
        handle_fpga_leds();
        usleep(ALT_MICROSECS_IN_A_SEC / 10);
    }

    munmap_peripherals();
    close_physical_memory_device();

    return 0;
}
```

Finally, in the main code, I first hardcode a base frequency to fpga_pio, and I also ask the code to activate the handle buttons function.

Here is a video demonstrating the workings of my homework:
https://youtu.be/DXvES32BrSk