Nome: Victor Dallagnol Bento e Victor Oliveira Costa

Curso: Engenharia de Computação

Objetivos: Em um primeiro momento o objetivo foi criar dois arquivos *.c* e *.h* com funções relacionadas ao acionamento de um LED e posteriormente com funções para portas programáveis de entrada e saída de dados (GPIO).

As funções criadas para as atividades estão contidas nos arquivos *.h* são as seguintes:

```
4   void led_config(uint8_t led_num, uint8_t pin);  // Configura Pino
5
6   void led_on(uint8_t led_num); // ligar
7
8   void led_off(uint8_t led_num); // desligar
9
10  void led_toggle(uint8_t led_num); // alternar
```

led-driver.h

```
1   void gpio_config(uint8_t gpio_num, uint8_t pin, uint8_t dir);
2
3   void gpio_write(uint8_t gpio_num, uint8_t val);
4
5   void gpio_read(uint8_t gpio_num, uint8_t *val);
6
7   void gpio_toggle(uint8_t gpio_num);
8
9   void gpio_dir(uint8_t gpio_num);
```

gpio-driver.h

Essas funções foram criadas e modificadas com base em funções já existentes, com o intuído de facilitar o entendimento no código, deixando os comandos mais similares com sua devida função.

Criou-se um vetor de pinos e de portas, visando deixar o código mais genérico, não somente para a FPGA utilizada. Esse vetor de pinos recebe o pino específico na posição do numero do LED. E o vetor de portas recebe o pino específico na posição do numero da porta. As funções criadas foram às seguintes:

```c
int pin_vec[30];

// Configuração dos PINOS e do LED
void led_config(uint8_t led_num, uint8_t pin){
    pin_vec[led_num] = pin;

    struct port_config pin_conf;
    port_get_config_defaults(&pin_conf);
    pin_conf.direction = PORT_PIN_DIR_OUTPUT;

    port_pin_set_config(pin_vec[led_num], &pin_conf);
}

// LIGAR
void led_on(uint8_t led_num){

    port_pin_set_output_level(pin_vec[led_num], LED_0_ACTIVE);
}

// DESLIGAR
void led_off(uint8_t led_num){

    port_pin_set_output_level(pin_vec[led_num], LED_0_INACTIVE);
}

// ALTERAR ESTADO
void led_toggle(uint8_t led_num){
    port_pin_toggle_output_level(pin_vec[led_num]);
}
```

led-driver.c

```c
int port_vec[30];
struct port_config config_pin[30];

// Configuração da estrutura do pino e inicialização da porta do pino/grupo de pinos
void porta_config(uint8_t port_num, uint8_t pin, uint8_t dir) {
    port_vec[port_num] = pin;

    port_get_config_defaults(&config_pin[port_num]);

    config_pin[port_num].direction  = dir;                  // Envia direção para port_num na estrutura
    port_pin_set_config(pin, &config_pin[port_num]);        // Escvreve num pino/porta de configuração
}

// Função para setar o estado da porta/pino
void porta_write(uint8_t port_num, uint8_t val) {
    if (val) {
        port_pin_set_output_level(port_vec[port_num], !true);
    } else {
        port_pin_set_output_level(port_vec[port_num], !false);
    }
}

// Lê(restaura) o estado da porta/pino
void porta_read(uint8_t port_num, uint8_t *val) {
    if (port_pin_get_output_level(port_vec[port_num])) {
        *val = 1;
    }
}

// Alternancia
void porta_toggle(uint8_t port_num) {
    port_pin_toggle_output_level(port_vec[port_num]);
}

// Direção
int porta_dir(uint8_t port_num) {
    int dir;
    dir = config_pin[port_num].direction;
}
```

gpio-driver.c

Em um primeiro momento efetuou-se os teste para as funções do LED. As funções contidas em *static void config_led(void)* foram postas nas novas funções criadas, e o primeiro teste foi efetuado, removendo a função antiga do código, permanecendo apenas com as novas funções criadas.

```
13  //~~~~~~~~~~~~~              FUNÇÃO ANTIGA       ~~~~~~~~~~~~~~~
14
15  /*static void config_led(void)
16  {
17      struct port_config pin_conf;
18      port_get_config_defaults(&pin_conf);
19
20      pin_conf.direction  = PORT_PIN_DIR_OUTPUT;
21      port_pin_set_config(LED_0_PIN, &pin_conf);
22      port_pin_set_output_level(LED_0_PIN, LED_0_INACTIVE);
23  }*/
24
25
26  int pin_vec[30];
27
28  // Configuração dos PINOS e do LED
29  void led_config(uint8_t led_num, uint8_t pin){
30      pin_vec[led_num] = pin;
31
32      struct port_config pin_conf;
33      port_get_config_defaults(&pin_conf);
34      pin_conf.direction = PORT_PIN_DIR_OUTPUT;
35
36      port_pin_set_config(pin_vec[led_num], &pin_conf);
37  }
38
39  // LIGAR
40  void led_on(uint8_t led_num){
41
42      port_pin_set_output_level(pin_vec[led_num], LED_0_ACTIVE);
43  }
44
45  // DESLIGAR
46  void led_off(uint8_t led_num){
47
48      port_pin_set_output_level(pin_vec[led_num], LED_0_INACTIVE);
49  }
50
51  // ALTERAR ESTADO
52  void led_toggle(uint8_t led_num){
53      port_pin_toggle_output_level(pin_vec[led_num]);
54  }
```

```
56  int main(void)
57  {
58      system_init();
59
60      /*Configure system tick to generate periodic interrupts */
61      //SysTick_Config(system_gclk_gen_get_hz(GCLK_GENERATOR_0));
62
63      led_config(0, LED_0_PIN);
64      led_on(0);
65      //led_off(0);
66      //led_toggle(0);
67
68      while (true) {
69      }
70  }
```

Em um segundo momento, adicionou-se o arquivo *.h* ao projeto, e os teste finais foram efetuados.

```c
#include <asf.h>
#include <led-driver.h>

/* void SysTick_Handler(void)
{
    port_pin_toggle_output_level(LED_0_PIN);
}*/

/** Configure LED0, turn it off*/

//~~~~~~~~~~~~~              FUNÇÃO ANTIGA       ~~~~~~~~~~~~~~~

/*static void config_led(void)
{
    struct port_config pin_conf;
    port_get_config_defaults(&pin_conf);

    pin_conf.direction  = PORT_PIN_DIR_OUTPUT;
    port_pin_set_config(LED_0_PIN, &pin_conf);
    port_pin_set_output_level(LED_0_PIN, LED_0_INACTIVE);
}*/


int pin_vec[30];

// Configuração dos PINOS e do LED
void led_config(uint8_t led_num, uint8_t pin){
    pin_vec[led_num] = pin;

    struct port_config pin_conf;
    port_get_config_defaults(&pin_conf);
    pin_conf.direction = PORT_PIN_DIR_OUTPUT;

    port_pin_set_config(pin_vec[led_num], &pin_conf);
}

// LIGAR
void led_on(uint8_t led_num){

    port_pin_set_output_level(pin_vec[led_num], LED_0_ACTIVE);
}
```

```c

// DESLIGAR
void led_off(uint8_t led_num){

    port_pin_set_output_level(pin_vec[led_num], LED_0_INACTIVE);
}

// ALTERAR ESTADO
void led_toggle(uint8_t led_num){
    port_pin_toggle_output_level(pin_vec[led_num]);
}

int main(void)
{
    system_init();

    /*Configure system tick to generate periodic interrupts */
    //SysTick_Config(system_gclk_gen_get_hz(GCLK_GENERATOR_0));

    led_config(0, LED_0_PIN);
    led_on(0);
    //led_off(0);
    //led_toggle(0);

    while (true) {
    }
}

```

C source file

Relacionado às funções de entrada e saída, percebeu-se que no próprio exemplo do LED existiam funções que relacionavam entradas e saídas. Como na atividade do LED, essas funções foram alteradas seguindo a proposta feita em aula e um novo projeto foi criado para testar as novas funções de GPIO.

```c
#include <asf.h>
#include <gpio-driver.h>

int port_vec[30];
struct port_config config_pin[30];

// Configuração da estrutura do pino e inicialização da porta do pino/grupo de pinos
void porta_config(uint8_t port_num, uint8_t pin, uint8_t dir) {
    port_vec[port_num] = pin;

    port_get_config_defaults(&config_pin[port_num]);

    config_pin[port_num].direction  = dir;                  // Envia direção para port_num na estrutura
    port_pin_set_config(pin, &config_pin[port_num]);        // Escvreve num pino/porta de configuração
}

// Função para setar o estado da porta/pino
void porta_write(uint8_t port_num, uint8_t val) {
    if (val) {
        port_pin_set_output_level(port_vec[port_num], !true);
    } else {
        port_pin_set_output_level(port_vec[port_num], !false);
    }
}

// Lê(restaura) o estado da porta/pino
void porta_read(uint8_t port_num, uint8_t *val) {
    if (port_pin_get_output_level(port_vec[port_num])) {
        *val = 1;
    }
}

// Alternancia
void porta_toggle(uint8_t port_num) {
    port_pin_toggle_output_level(port_vec[port_num]);
}

// Direção
int porta_dir(uint8_t port_num) {
    int dir;
    dir = config_pin[port_num].direction;
}
```

```c
int main(void)
{
    system_init();

    /*Configure system tick to generate periodic interrupts */
    //SysTick_Config(system_gclk_gen_get_hz(GCLK_GENERATOR_0));

    int led = 1;
    int t = 0;

    porta_config(led, LED_0_PIN, PORT_PIN_DIR_OUTPUT);

    porta_write(led, 1);

    while (true) {
        if (t == 100000) {
            porta_toggle(led);
            t = 0;
        } else
            t++;
    }
}
```

Para efetuar o teste adicionou-se no laço (*whille*) um trecho de código, com condições de tempo, para que a porta do LED pudesse ser alterada.