

## Assignment 5- Implementing a closed loop control application in Zephyr V1.0

Generated by Doxygen 1.8.17



<b>1 Bug List</b>	<b>1</b>
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 File Documentation</b>	<b>5</b>
3.1 CMakeLists.txt File Reference	5
3.1.1 Function Documentation	5
3.1.1.1 cmake_minimum_required()	5
3.2 main.c File Reference	5
3.2.1 Macro Definition Documentation	7
3.2.1.1 ADC_ACQUISITION_TIME	7
3.2.1.2 ADC_CHANNEL_ID	8
3.2.1.3 ADC_CHANNEL_INPUT	8
3.2.1.4 ADC_GAIN	8
3.2.1.5 ADC_NID	8
3.2.1.6 ADC_REFERENCE	8
3.2.1.7 ADC_RESOLUTION	8
3.2.1.8 BOARDBUT1	8
3.2.1.9 BOARDBUT2 [1/2]	9
3.2.1.10 BOARDBUT2 [2/2]	9
3.2.1.11 BOARDBUT4	9
3.2.1.12 BOARDLED1	9
3.2.1.13 BUFFER_SIZE	9
3.2.1.14 FATAL_ERR	9
3.2.1.15 GPIO0_NID	9
3.2.1.16 len_dados	9
3.2.1.17 MAIN_SLEEP_TIME_MS	10
3.2.1.18 PWM0_NID	10
3.2.1.19 RX_TIMEOUT	10
3.2.1.20 RXBUF_SIZE	10
3.2.1.21 SAMP_PERIOD_MS	10
3.2.1.22 STACK_SIZE	10
3.2.1.23 thread_A1_prio	10
3.2.1.24 thread_A_prio	11
3.2.1.25 thread_B_prio	11
3.2.1.26 thread_C_prio	11
3.2.1.27 thread_D_prio	11
3.2.1.28 TXBUF_SIZE	11
3.2.1.29 UART_NID	11
3.2.2 Function Documentation	11
3.2.2.1 but1press_cbfunction()	11
3.2.2.2 but2press_cbfunction()	12

3.2.2.3 but4press_cbfunction()	12
3.2.2.4 K_THREAD_STACK_DEFINE() [1/5]	13
3.2.2.5 K_THREAD_STACK_DEFINE() [2/5]	13
3.2.2.6 K_THREAD_STACK_DEFINE() [3/5]	13
3.2.2.7 K_THREAD_STACK_DEFINE() [4/5]	13
3.2.2.8 K_THREAD_STACK_DEFINE() [5/5]	13
3.2.2.9 main()	14
3.2.2.10 thread_A1_code()	15
3.2.2.11 thread_A_code()	15
3.2.2.12 thread_B_code()	16
3.2.2.13 thread_C_code()	17
3.2.2.14 thread_D_code()	18
3.2.3 Variable Documentation	19
3.2.3.1 a1a	19
3.2.3.2 ab	19
3.2.3.3 adc_dev	19
3.2.3.4 bc	19
3.2.3.5 bd	19
3.2.3.6 Flag_1	20
3.2.3.7 Flag_2	20
3.2.3.8 Flag_3	20
3.2.3.9 Flag_4	20
3.2.3.10 flag_flag	20
3.2.3.11 gpio0_dev	20
3.2.3.12 my_timer	20
3.2.3.13 rep_mesg	20
3.2.3.14 sem_a1a	21
3.2.3.15 sem_ab	21
3.2.3.16 sem_bc	21
3.2.3.17 sem_bd	21
3.2.3.18 thread_A1_data	21
3.2.3.19 thread_A1_tid	21
3.2.3.20 thread_A_data	21
3.2.3.21 thread_A_tid	21
3.2.3.22 thread_B_data	22
3.2.3.23 thread_B_tid	22
3.2.3.24 thread_C_data	22
3.2.3.25 thread_C_tid	22
3.2.3.26 thread_D_data	22
3.2.3.27 thread_D_tid	22
3.2.3.28 uart_cfg	22
3.2.3.29 uart_dev	23

---

3.2.3.30 uart_rx_rdy_flag . . . . .	23
3.2.3.31 welcome_mesg . . . . .	23
3.3 main.h File Reference . . . . .	23
3.3.1 Detailed Description . . . . .	24
3.3.2 Function Documentation . . . . .	24
3.3.2.1 but1press_cbfunction() . . . . .	24
3.3.2.2 but2press_cbfunction() . . . . .	24
3.3.2.3 but4press_cbfunction() . . . . .	25
3.3.2.4 main() . . . . .	25
3.3.2.5 thread_A1_code() . . . . .	26
3.3.2.6 thread_A_code() . . . . .	27
3.3.2.7 thread_B_code() . . . . .	28
3.3.2.8 thread_C_code() . . . . .	29
3.3.2.9 thread_D_code() . . . . .	30
<b>Index</b>	<b>33</b>



## Chapter 1

# Bug List

File [main.h](#)

No known bugs.





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.c</a>	.....	5
<a href="#">main.h</a>		
	Application to control the light intensity of a given region. The system comprises a light sensor, an illumination system and a Human-Machine Interface. The system can operate in two modes↔ : • Automatic: programmable via the terminal. Should allow setting On/Off periods and the corresponding light intensity; • Manual: interface via the DevKit buttons. Allows to turn the system On/Off (when in “Off” the system operates in automatic mode), via two of the buttons. The other two buttons allow to set (increase/decrease) the desired intensity	23



## Chapter 3

# File Documentation

### 3.1 CMakeLists.txt File Reference

#### Functions

- [cmake\\_minimum\\_required](#) (VERSION 3.20.0) find\_package(Zephyr REQUIRED HINTS \$ENV

#### 3.1.1 Function Documentation

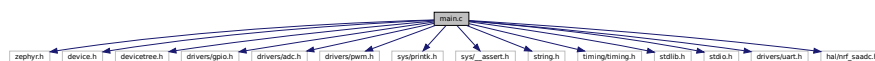
##### 3.1.1.1 cmake\_minimum\_required()

```
cmake_minimum_required (
    VERSION 3.20.  0 )
```

### 3.2 main.c File Reference

```
#include <zephyr.h>
#include <device.h>
#include <devicetree.h>
#include <drivers/gpio.h>
#include <drivers/adc.h>
#include <drivers/pwm.h>
#include <sys/printk.h>
#include <sys/__assert.h>
#include <string.h>
#include <timing/timing.h>
#include <stdlib.h>
#include <stdio.h>
#include <drivers/uart.h>
#include <hal/nrf_saadc.h>
```

Include dependency graph for main.c:



## Macros

- `#define len_dados 10`
- `#define STACK_SIZE 1024`
- `#define thread_A1_prio 1`
- `#define thread_A_prio 1`
- `#define thread_B_prio 1`
- `#define thread_C_prio 1`
- `#define thread_D_prio 1`
- `#define SAMP_PERIOD_MS 1000`
- `#define ADC_NID DT_NODELABEL(adc)`
- `#define ADC_RESOLUTION 10`
- `#define ADC_GAIN ADC_GAIN_1_4`
- `#define ADC_REFERENCE ADC_REF_VDD_1_4`
- `#define ADC_ACQUISITION_TIME ADC_ACQ_TIME(ADC_ACQ_TIME_MICROSECONDS, 40)`
- `#define ADC_CHANNEL_ID 1`
- `#define ADC_CHANNEL_INPUT NRF_SAADC_INPUT_AIN1`
- `#define BUFFER_SIZE 1`
- `#define GPIO0_NID DT_NODELABEL(gpio0)`
- `#define PWM0_NID DT_NODELABEL(pwm0)`
- `#define BOARDLED1 0x0d`
- `#define BOARDBUT1 0xb /* Pin at which BUT1 is connected. Addressing is direct (i.e., pin number) */`
- `#define BOARDBUT2 0xc`
- `#define BOARDBUT2 0xc`
- `#define BOARDBUT4 0x19`
- `#define FATAL_ERR -1 /* Fatal error return code, app terminates */`
- `#define UART_NID DT_NODELABEL(uart0) /* UART Node label, see dts */`
- `#define RXBUF_SIZE 60 /* RX buffer size */`
- `#define TXBUF_SIZE 60 /* TX buffer size */`
- `#define RX_TIMEOUT 1000 /* Inactivity period after the instant when last char was received that triggers an rx event (in us) */`
- `#define MAIN_SLEEP_TIME_MS 10 /* Time between main() activations */`

## Functions

- `K_THREAD_STACK_DEFINE` (thread\_A1\_stack, `STACK_SIZE`)
- `K_THREAD_STACK_DEFINE` (thread\_A\_stack, `STACK_SIZE`)
- `K_THREAD_STACK_DEFINE` (thread\_B\_stack, `STACK_SIZE`)
- `K_THREAD_STACK_DEFINE` (thread\_C\_stack, `STACK_SIZE`)
- `K_THREAD_STACK_DEFINE` (thread\_D\_stack, `STACK_SIZE`)
- `void thread_A1_code` (void \*argA, void \*argB, void \*argC)
 

*É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por "ab" e no final faz give do semáforo AB.*
- `void thread_A_code` (void \*argA, void \*argB, void \*argC)
 

*É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por "ab" e no final faz give do semáforo AB.*
- `void thread_B_code` (void \*argA, void \*argB, void \*argC)
 

*é feito o take do semáforo AB é realizado uma média das últimas 10 amostras calculadas na thread A e é feito um filtro rejeitando todos os valores que estejam abaixo ou acima de 10% da média, sendo que este output é colocado numa variável global (shared memory between tasks B/C) no nosso Código denominada por "cb" e no final faz give do semáforo BC.*
- `void thread_C_code` (void \*argA, void \*argB, void \*argC)
- `void thread_D_code` (void \*argA, void \*argB, void \*argC)
 

*Takes one adc\_sample.*

- void [but1press\\_cbfunction](#) (const struct device \*dev, struct gpio\_callback \*cb, uint32\_t pins)  
*If button 1 is pressed, Update Flag 1.*
- void [but2press\\_cbfunction](#) (const struct device \*dev, struct gpio\_callback \*cb, uint32\_t pins)  
*If button 2 is pressed, Update Flag 2.*
- void [but4press\\_cbfunction](#) (const struct device \*dev, struct gpio\_callback \*cb, uint32\_t pins)  
*If button 4 is pressed, Update Flag 4.*
- void [main](#) (void)  
*Main funtion: Initialize semaphores and configure GPIO\_PIN (UART)*

## Variables

- struct k\_thread [thread\\_A1\\_data](#)
- struct k\_thread [thread\\_A\\_data](#)
- struct k\_thread [thread\\_B\\_data](#)
- struct k\_thread [thread\\_C\\_data](#)
- struct k\_thread [thread\\_D\\_data](#)
- k\_tid\_t [thread\\_A1\\_tid](#)
- k\_tid\_t [thread\\_A\\_tid](#)
- k\_tid\_t [thread\\_B\\_tid](#)
- k\_tid\_t [thread\\_C\\_tid](#)
- k\_tid\_t [thread\\_D\\_tid](#)
- int [a1a](#) = 0
- int [ab](#) = 0
- int [bc](#) = 0
- int [bd](#) = 0
- struct k\_sem [sem\\_a1a](#)
- struct k\_sem [sem\\_ab](#)
- struct k\_sem [sem\\_bc](#)
- struct k\_sem [sem\\_bd](#)
- struct k\_timer [my\\_timer](#)
- const struct device \* [adc\\_dev](#) = NULL
- volatile int [Flag\\_1](#) = 0
- volatile int [Flag\\_3](#) = 0
- volatile int [Flag\\_2](#) = 0
- volatile int [Flag\\_4](#) = 0
- volatile bool [flag\\_flag](#) = 0
- const struct device \* [gpio0\\_dev](#)
- const struct uart\_config [uart\\_cfg](#)
- const struct device \* [uart\\_dev](#)
- volatile int [uart\\_rx\\_rdy\\_flag](#)
- uint8\_t [welcome\\_mesg](#) [] = "UART demo: Type a few chars in a row and then pause for a little while ...\n\r"
- uint8\_t [rep\\_mesg](#) [TXBUF\_SIZE]

## 3.2.1 Macro Definition Documentation

### 3.2.1.1 ADC\_ACQUISITION\_TIME

```
#define ADC_ACQUISITION_TIME ADC_ACQ_TIME(ADC_ACQ_TIME_MICROSECONDS, 40)
```

### 3.2.1.2 ADC\_CHANNEL\_ID

```
#define ADC_CHANNEL_ID 1
```

### 3.2.1.3 ADC\_CHANNEL\_INPUT

```
#define ADC_CHANNEL_INPUT NRF_SAADC_INPUT_AIN1
```

### 3.2.1.4 ADC\_GAIN

```
#define ADC_GAIN ADC_GAIN_1_4
```

### 3.2.1.5 ADC\_NID

```
#define ADC_NID DT_NODELABEL(adc)
```

ADC definitions and includes

### 3.2.1.6 ADC\_REFERENCE

```
#define ADC_REFERENCE ADC_REF_VDD_1_4
```

### 3.2.1.7 ADC\_RESOLUTION

```
#define ADC_RESOLUTION 10
```

### 3.2.1.8 BOARDBUT1

```
#define BOARDBUT1 0xb /* Pin at which BUT1 is connected. Addressing is direct (i.e., pin  
number) */
```

### 3.2.1.9 BOARD BUT2 [1/2]

```
#define BOARD BUT2 0xc
```

### 3.2.1.10 BOARD BUT2 [2/2]

```
#define BOARD BUT2 0xc
```

### 3.2.1.11 BOARD BUT4

```
#define BOARD BUT4 0x19
```

### 3.2.1.12 BOARD LED1

```
#define BOARD LED1 0x0d
```

### 3.2.1.13 BUFFER\_SIZE

```
#define BUFFER_SIZE 1
```

### 3.2.1.14 FATAL\_ERR

```
#define FATAL_ERR -1 /* Fatal error return code, app terminates */
```

### 3.2.1.15 GPIO0\_NID

```
#define GPIO0_NID DT_NODELABEL(gpio0)
```

Refer to dts file

### 3.2.1.16 len\_dados

```
#define len_dados 10
```

Number of samples for the average

### 3.2.1.17 MAIN\_SLEEP\_TIME\_MS

```
#define MAIN_SLEEP_TIME_MS 10 /* Time between main() activations */
```

### 3.2.1.18 PWM0\_NID

```
#define PWM0_NID DT_NODELABEL(pwm0)
```

### 3.2.1.19 RX\_TIMEOUT

```
#define RX_TIMEOUT 1000 /* Inactivity period after the instant when last char was received  
that triggers an rx event (in us) */
```

### 3.2.1.20 RXBUF\_SIZE

```
#define RXBUF_SIZE 60 /* RX buffer size */
```

### 3.2.1.21 SAMP\_PERIOD\_MS

```
#define SAMP_PERIOD_MS 1000
```

Therad periodicity (in ms)

### 3.2.1.22 STACK\_SIZE

```
#define STACK_SIZE 1024
```

Size of stack area used by each thread (can be thread specific, if necessary)

### 3.2.1.23 thread\_A1\_prio

```
#define thread_A1_prio 1
```

Thread scheduling priority



### 3.2.1.24 thread\_A\_prio

```
#define thread_A_prio 1
```

### 3.2.1.25 thread\_B\_prio

```
#define thread_B_prio 1
```

### 3.2.1.26 thread\_C\_prio

```
#define thread_C_prio 1
```

### 3.2.1.27 thread\_D\_prio

```
#define thread_D_prio 1
```

### 3.2.1.28 TXBUF\_SIZE

```
#define TXBUF_SIZE 60 /* TX buffer size */
```

### 3.2.1.29 UART\_NID

```
#define UART_NID DT_NODELABEL(uart0) /* UART Node label, see dts */
```

## 3.2.2 Function Documentation

### 3.2.2.1 but1press\_cbfunction()

```
void but1press_cbfunction (
    const struct device * dev,
    struct gpio_callback * cb,
    uint32_t pins )
```

If button 1 is pressed, Update Flag 1.

```
void but1press_cbfunction(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
{
    Flag_1 = 1;
    flag_flag=!flag_flag;
    printk("but1 \n\r");
}
```

**Parameters**

<i>arg3</i>	const struct device *dev, struct gpio_callback *cb, uint32_t pins.
-------------	--

**Returns**

No returns

**3.2.2.2 but2press\_cbfunction()**

```
void but2press_cbfunction (
    const struct device * dev,
    struct gpio_callback * cb,
    uint32_t pins )
```

If button 2 is pressed, Update Flag 2.

```
void but1press_cbfunction(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
{
    Flag_2 = 1;
}
```

**Parameters**

<i>arg3</i>	const struct device *dev, struct gpio_callback *cb, uint32_t pins.
-------------	--

**Returns**

No returns

**3.2.2.3 but4press\_cbfunction()**

```
void but4press_cbfunction (
    const struct device * dev,
    struct gpio_callback * cb,
    uint32_t pins )
```

If button 4 is pressed, Update Flag 4.

```
void but1press_cbfunction(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
{
    Flag_4 = 1;
}
```

**Parameters**

<i>arg3</i>	const struct device *dev, struct gpio_callback *cb, uint32_t pins.
-------------	--

## Returns

No returns

### 3.2.2.4 K\_THREAD\_STACK\_DEFINE() [1/5]

```
K_THREAD_STACK_DEFINE (
    thread_A1_stack ,
    STACK_SIZE )
```

Create thread stack space

### 3.2.2.5 K\_THREAD\_STACK\_DEFINE() [2/5]

```
K_THREAD_STACK_DEFINE (
    thread_A_stack ,
    STACK_SIZE )
```

### 3.2.2.6 K\_THREAD\_STACK\_DEFINE() [3/5]

```
K_THREAD_STACK_DEFINE (
    thread_B_stack ,
    STACK_SIZE )
```

### 3.2.2.7 K\_THREAD\_STACK\_DEFINE() [4/5]

```
K_THREAD_STACK_DEFINE (
    thread_C_stack ,
    STACK_SIZE )
```

### 3.2.2.8 K\_THREAD\_STACK\_DEFINE() [5/5]

```
K_THREAD_STACK_DEFINE (
    thread_D_stack ,
    STACK_SIZE )
```

### 3.2.2.9 main()

```
void main (
    void )
```

Main funtion: Initialize semaphores and configure GPIO\_PIN (UART)

```
void main(void) {
    int err=0;
    printf("\n\n Illustration of the use of shmem + semaphores\n\n");
    int ret = 0;
    ret = gpio_pin_configure(gpio0_dev, BOARDBUT1, GPIO_INPUT | GPIO_PULL_UP);
    ret = gpio_pin_configure(gpio0_dev, BOARDBUT2, GPIO_INPUT | GPIO_PULL_UP);
    ret = gpio_pin_configure(gpio0_dev, BOARDBUT4, GPIO_INPUT | GPIO_PULL_UP);
    if (ret < 0) {
        printk("Error %d: Failed to configure BUT 1 \r", ret);
        return;
    }
    ret = gpio_pin_interrupt_configure(gpio0_dev, BOARDBUT1, GPIO_INT_EDGE_TO_ACTIVE);
    ret = gpio_pin_interrupt_configure(gpio0_dev, BOARDBUT2, GPIO_INT_EDGE_TO_ACTIVE);
    ret = gpio_pin_interrupt_configure(gpio0_dev, BOARDBUT4, GPIO_INT_EDGE_TO_ACTIVE);
    if (ret != 0) {
        printk("Error %d: failed to configure interrupt on BUT1 pin \r", ret);
        return;
    }
    gpio_init_callback(&but1_cb_data, but1press_cbfunction, BIT(BOARDBUT1));
    gpio_add_callback(gpio0_dev, &but1_cb_data);
    gpio_init_callback(&but2_cb_data, but2press_cbfunction, BIT(BOARDBUT2));
    gpio_add_callback(gpio0_dev, &but2_cb_data);
    gpio_init_callback(&but4_cb_data, but4press_cbfunction, BIT(BOARDBUT4));
    gpio_add_callback(gpio0_dev, &but4_cb_data);

    err=0;
    uint8_t welcome_mesg[] = "UART demo: Type a few chars in a row and then pause for a little while
        ... \n\n\r";
    uint8_t rep_mesg[TXBUF_SIZE];

    k_sem_init(&sem_ala, 0, 1);
    k_sem_init(&sem_ab, 0, 1);
    k_sem_init(&sem_bc, 0, 1);
    k_sem_init(&sem_bd, 0, 1);

    thread_A1_tid = k_thread_create(&thread_A1_data, thread_A1_stack,
        K_THREAD_STACK_SIZEOF(thread_A1_stack), thread_A1_code,
        NULL, NULL, NULL, thread_A1_prio, 0, K_NO_WAIT);
    thread_A_tid = k_thread_create(&thread_A_data, thread_A_stack,
        K_THREAD_STACK_SIZEOF(thread_A_stack), thread_A_code,
        NULL, NULL, NULL, thread_A_prio, 0, K_NO_WAIT);
    thread_B_tid = k_thread_create(&thread_B_data, thread_B_stack,
        K_THREAD_STACK_SIZEOF(thread_B_stack), thread_B_code,
        NULL, NULL, NULL, thread_B_prio, 0, K_NO_WAIT);
    thread_C_tid = k_thread_create(&thread_C_data, thread_C_stack,
        K_THREAD_STACK_SIZEOF(thread_C_stack), thread_C_code,
        NULL, NULL, NULL, thread_C_prio, 0, K_NO_WAIT);
    thread_D_tid = k_thread_create(&thread_D_data, thread_D_stack,
        K_THREAD_STACK_SIZEOF(thread_D_stack), thread_D_code,
        NULL, NULL, NULL, thread_D_prio, 0, K_NO_WAIT);

    return;
}
```

#### Parameters

<i>NO_args</i>	without arguments
----------------	-------------------

#### Returns

No returns

Welcome message

Create and init semaphores

Create tasks

## 3.2.2.10 thread\_A1\_code()

```
void thread_A1_code (
    void * argA,
    void * argB,
    void * argC )
```

É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por “ab” e no final faz give do semáforo AB.

Thread code prototypes

## 3.2.2.11 thread\_A\_code()

```
void thread_A_code (
    void * argA,
    void * argB,
    void * argC )
```

É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por “ab” e no final faz give do semáforo AB.

```
void thread_A_code(void *argA , void *argB, void *argC)
{
    int err=0;

    printk("Thread A init\n");

    adc_dev = device_get_binding(DT_LABEL(ADC_NID));
    if (!adc_dev) {
        printk("ADC device_get_binding() failed\n");
    }
    err = adc_channel_setup(adc_dev, &my_channel_cfg);
    if (err) {
        printk("adc_channel_setup() failed with error code %d\n", err);
    }

    NRF_SAADC->TASKS_CALIBRATEOFFSET = 1;
    while(1) {
        k_sem_take(&sem_a1a, K_FOREVER);

        err=adc_sample();
        if(err) {
            printk("adc_sample() failed with error code %d\n\r",err);
        }
        else {
            if(adc_sample_buffer[0] > 1023) {
                printk("adc reading out of range\n\r");
            }
            else {
                ab=adc_sample_buffer[0];
            }
        }
        printk("Thread A set ab value to: %d ",ab);

        k_sem_give(&sem_ab);
    }
}
```

## Parameters

arg3	void *argA , void *argB, void *argC.
------	--------------------------------------

## Returns

No returns

## 3.2.2.12 thread\_B\_code()

```
void thread_B_code (
    void * argA,
    void * argB,
    void * argC )
```

é feito o take do semáforo AB é realizado uma média das últimas 10 amostras calculadas na thread A e é feito um filtro rejeitando todos os valores que estejam abaixo ou acima de 10% da média, sendo que este output é colocado numa variável global (shared memory between tasks B/C) no nosso Código denominada por “cb” e no final faz give do semáforo BC.

```
void thread_B_code(void *argA , void *argB, void *argC)
{
    int Array_dados[len_dados]={0};
    int k=0;
    printk("Thread B init (sporadic, waits on a semaphore by task A)\n");
    while(1) {
        int sumador=0,somador_2=0,media=0, media_filtered=0;
        int contador=0;

        k_sem_take(&sem_ab, K_FOREVER);

        printk("Task B read ab value: %d\n",ab);
        for(int k=len_dados-1; k>0;k--){

            Array_dados[k]= Array_dados[k-1];
        }
        Array_dados[0]= ab;

        for(int i = 0; i < len_dados; i++){
            if(Array_dados[i] != 0){
                sumador = sumador + Array_dados[i];
            }
        }
        media=sumador/len_dados;
        contador=0;

        for(int j = 0; j < len_dados; j++){
            if(Array_dados[j] < (media - media*0.1) || Array_dados[j] > (media + media*0.1))
                somador_2=somador_2;
            else{
                somador_2 = somador_2 + Array_dados[j];
                contador =contador +1;
            }
        }
        if(somador_2 != 0)
            media_filtered=somador_2/contador;
        else
            media_filtered = 0;
        bd=ab;
        printk("Thread B set bc value to: %d\n",bc);
        k_sem_give(&sem_bd);
    }
}
```

## Parameters

<i>arg3</i>	void *argA , void *argB, void *argC.
-------------	--------------------------------------

## Returns

No returns

array de dados da adc

### 3.2.2.13 thread\_C\_code()

```
void thread_C_code (
    void * argA,
    void * argB,
    void * argC )
void thread_C_code(void *argA , void *argB, void *argC)
{
    const struct device *gpio0_dev;
    const struct device *pwm0_dev;
    int ret=0;
    unsigned int dcValue[]={100,90,80,70,60,50,40,30,20,10,0};
    unsigned int dcIndex=0;
    unsigned int pwmPeriod_us = 100;
    printk("Thread C init (sporadic, waits on a semaphore by task B)\n");

    gpio0_dev = device_get_binding(DT_LABEL(GPIO0_NID));
    if (gpio0_dev == NULL) {
        printk("Error: Failed to bind to GPIO0\n\r");
        return;
    }

    pwm0_dev = device_get_binding(DT_LABEL(PWM0_NID));
    if (pwm0_dev == NULL) {
        printk("Error: Failed to bind to PWM0\n r");
        return;
    }

    while(1) {
        k_sem_take(&sem_bc, K_FOREVER);
        ret=0;

        if(Flag_2) {
            dcIndex++;
            if(dcIndex == 11)
                dcIndex = 0;
            Flag_2 = 0;
            printk("PWM DC value set to %u %%\n\r",dcValue[dcIndex]);
            ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
                pwmPeriod_us, (unsigned int)((pwmPeriod_us*dcValue[dcIndex])/100), PWM_POLARITY_NORMAL);
            if (ret) {
                printk("Error %d: failed to set pulse width\n", ret);
            }
            return;
        }

        if(Flag_4) {
            if(dcIndex == 0)
                dcIndex = 11;
            dcIndex--;
            Flag_4 = 0;
            printk("PWM DC value set to %u %%\n\r",dcValue[dcIndex]);
            ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
                pwmPeriod_us, (unsigned int)((pwmPeriod_us*dcValue[dcIndex])/100), PWM_POLARITY_NORMAL);
            if (ret) {
                printk("Error %d: failed to set pulse width\n", ret);
            }
            return;
        }

        printk("Task C - PWM: %u % n", (unsigned int)((pwmPeriod_us*bc)/1023)/10));
    }
}
```

#### Parameters

<i>arg3</i>	void *argA , void *argB, void *argC.
-------------	--------------------------------------

#### Returns

No returns

Prints dutty-cycle

### 3.2.2.14 thread\_D\_code()

```
void thread_D_code (
    void * argA,
    void * argB,
    void * argC )
```

Takes one `adc_sample`.

```
void thread_D_code(void *argA , void *argB, void *argC)
{
    const struct device *gpio0_dev;
    const struct device *pwm0_dev;
    int ret=0;
    unsigned int dcValue[]={100,90,80,70,60,50,40,30,20,10,0};
    unsigned int dcIndex=0;
    unsigned int pwmPeriod_us = 100;
    printk("Thread C init (sporadic, waits on a semaphore by task B)\n");

    gpio0_dev = device_get_binding(DT_LABEL(GPIO0_NID));
    if (gpio0_dev == NULL) {
        printk("Error: Failed to bind to GPIO0\n r");
        return;
    }

    pwm0_dev = device_get_binding(DT_LABEL(PWM0_NID));
    if (pwm0_dev == NULL) {
        printk("Error: Failed to bind to PWM0\n r");
        return;
    }

    while(1) {
        k_sem_take(&sem_bd, K_FOREVER);
        printk("Valor lido para automatico %d\n r",bd);
        ret=0;

        if(bd<500) {

            ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
                pwmPeriod_us, (unsigned int) (pwmPeriod_us), PWM_POLARITY_NORMAL);
            if (ret) {
                printk("Error %d: failed to set pulse width\n", ret);
                return;
            }
        }
        else {

            ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
                pwmPeriod_us, (unsigned int) (0), PWM_POLARITY_NORMAL);
            if (ret) {
                printk("Error %d: failed to set pulse width\n", ret);
                return;
            }
        }
    }
}
```

#### Parameters

<i>NO_args</i>	without arguments
<i>arg3</i>	void *argA , void *argB, void *argC.

```
static int adc_sample(void)
{
    int ret;
    const struct adc_sequence sequence = {
        .channels = BIT(ADC_CHANNEL_ID),
        .buffer = adc_sample_buffer,
        .buffer_size = sizeof(adc_sample_buffer),
        .resolution = ADC_RESOLUTION,
    };
    if (adc_dev == NULL) {
        printk("adc_sample(): error, must bind to adc first \r");
        return -1;
    }
    ret = adc_read(adc_dev, &sequence);
    if (ret) {
        printk("adc_read() failed with code %d\n", ret);
    }
}
```



```
    }  
    return ret;  
}
```

#### Parameters

<i>NO_args</i>	without arguments
----------------	-------------------

#### Returns

Read ADC\_sample value (static int)

### 3.2.3 Variable Documentation

#### 3.2.3.1 a1a

```
int a1a =0
```

Global vars (shared memory between tasks A/B and B/C, resp)

#### 3.2.3.2 ab

```
int ab = 0
```

#### 3.2.3.3 adc\_dev

```
const struct device* adc_dev = NULL
```

#### 3.2.3.4 bc

```
int bc = 0
```

#### 3.2.3.5 bd

```
int bd = 0
```

#### 3.2.3.6 Flag\_1

```
volatile int Flag_1 = 0
```

#### 3.2.3.7 Flag\_2

```
volatile int Flag_2 = 0
```

#### 3.2.3.8 Flag\_3

```
volatile int Flag_3 = 0
```

#### 3.2.3.9 Flag\_4

```
volatile int Flag_4 = 0
```

#### 3.2.3.10 flag\_flag

```
volatile bool flag_flag = 0
```

#### 3.2.3.11 gpio0\_dev

```
const struct device* gpio0_dev
```

#### 3.2.3.12 my\_timer

```
struct k_timer my_timer
```

Global vars

#### 3.2.3.13 rep\_mesg

```
uint8_t rep_mesg[TXBUF\_SIZE]
```

#### 3.2.3.14 sem\_a1a

```
struct k_sem sem_a1a
```

Semaphores for task synch

#### 3.2.3.15 sem\_ab

```
struct k_sem sem_ab
```

#### 3.2.3.16 sem\_bc

```
struct k_sem sem_bc
```

#### 3.2.3.17 sem\_bd

```
struct k_sem sem_bd
```

#### 3.2.3.18 thread\_A1\_data

```
struct k_thread thread_A1_data
```

Create variables for thread data

#### 3.2.3.19 thread\_A1\_tid

```
k_tid_t thread_A1_tid
```

Create task IDs

#### 3.2.3.20 thread\_A\_data

```
struct k_thread thread_A_data
```

#### 3.2.3.21 thread\_A\_tid

```
k_tid_t thread_A_tid
```

### 3.2.3.22 thread\_B\_data

```
struct k_thread thread_B_data
```

### 3.2.3.23 thread\_B\_tid

```
k_tid_t thread_B_tid
```

### 3.2.3.24 thread\_C\_data

```
struct k_thread thread_C_data
```

### 3.2.3.25 thread\_C\_tid

```
k_tid_t thread_C_tid
```

### 3.2.3.26 thread\_D\_data

```
struct k_thread thread_D_data
```

### 3.2.3.27 thread\_D\_tid

```
k_tid_t thread_D_tid
```

### 3.2.3.28 uart\_cfg

```
const struct uart_config uart_cfg
```

#### Initial value:

```
= {  
    .baudrate = 115200,  
    .parity = UART_CFG_PARITY_NONE,  
    .stop_bits = UART_CFG_STOP_BITS_1,  
    .data_bits = UART_CFG_DATA_BITS_8,  
    .flow_ctrl = UART_CFG_FLOW_CTRL_NONE  
}
```

### 3.2.3.29 uart\_dev

```
const struct device* uart_dev
```

### 3.2.3.30 uart\_rx\_rdy\_flag

```
volatile int uart_rx_rdy_flag
```

### 3.2.3.31 welcome\_mesg

```
uint8_t welcome_mesg[] = "UART demo: Type a few chars in a row and then pause for a little  
while ...\n\r"
```

Main function

## 3.3 main.h File Reference

Application to control the light intensity of a given region. The system comprises a light sensor, an illumination system and a Human-Machine Interface. The system can operate in two modes: • Automatic: programmable via the terminal. Should allow setting On/Off periods and the corresponding light intensity; • Manual: interface via the DevKit buttons. Allows to turn the system On/Off (when in “Off” the system operates in automatic mode), via two of the buttons. The other two buttons allow to set (increase/decrease) the desired intensity.

### Functions

- void [main](#) (void)  
*Main funtion: Initialize semaphores and configure GPIO\_PIN (UART)*
- void [thread\\_A\\_code](#) (void \*argA, void \*argB, void \*argC)  
*É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por “ab” e no final faz give do semáforo AB.*
- void [thread\\_A1\\_code](#) (void \*argA, void \*argB, void \*argC)  
*É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por “ab” e no final faz give do semáforo AB.*
- void [thread\\_B\\_code](#) (void \*argA, void \*argB, void \*argC)  
*é feito o take do semáforo AB é realizado uma média das últimas 10 amostras calculadas na thread A e é feito um filtro rejeitando todos os valores que estejam abaixo ou acima de 10% da média, sendo que este output é colocado numa variável global (shared memory between tasks B/C) no nosso Código denominada por “cb” e no final faz give do semáforo BC.*
- void [thread\\_C\\_code](#) (void \*argA, void \*argB, void \*argC)
- void [thread\\_D\\_code](#) (void \*argA, void \*argB, void \*argC) static int adc\_sample(void)  
*Takes one adc\_sample.*
- void [but1press\\_cbfunction](#) (const struct device \*dev, struct gpio\_callback \*cb, uint32\_t pins)  
*If button 1 is pressed, Update Flag 1.*
- void [but2press\\_cbfunction](#) (const struct device \*dev, struct gpio\_callback \*cb, uint32\_t pins)  
*If button 2 is pressed, Update Flag 2.*
- void [but4press\\_cbfunction](#) (const struct device \*dev, struct gpio\_callback \*cb, uint32\_t pins)  
*If button 4 is pressed, Update Flag 4.*

### 3.3.1 Detailed Description

Application to control the light intensity of a given region. The system comprises a light sensor, an illumination system and a Human-Machine Interface. The system can operate in two modes: • Automatic: programmable via the terminal. Should allow setting On/Off periods and the corresponding light intensity; • Manual: interface via the DevKit buttons. Allows to turn the system On/Off (when in “Off” the system operates in automatic mode), via two of the buttons. The other two buttons allow to set (increase/decrease) the desired intensity.

#### Author

Frederico Moreira, Ana Sousa, Pedro Rodrigues

#### Date

21 June 2022

**Bug** No known bugs.

### 3.3.2 Function Documentation

#### 3.3.2.1 but1press\_cbfunction()

```
void but1press_cbfunction (
    const struct device * dev,
    struct gpio_callback * cb,
    uint32_t pins )
```

If button 1 is pressed, Update Flag 1.

```
void but1press_cbfunction(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
{
    Flag_1 = 1;
    flag_flag=!flag_flag;
    printk("but1 \n\r");
}
```

#### Parameters

<i>arg3</i>	const struct device *dev, struct gpio_callback *cb, uint32_t pins.
-------------	--

#### Returns

No returns

#### 3.3.2.2 but2press\_cbfunction()

```
void but2press_cbfunction (
    const struct device * dev,
```

```

    struct gpio_callback * cb,
    uint32_t pins )

```

If button 2 is pressed, Update Flag 2.

```

void but1press_cbfunction(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
{
    Flag_2 = 1;
}

```

#### Parameters

<i>arg3</i>	const struct device *dev, struct gpio_callback *cb, uint32_t pins.
-------------	--

#### Returns

No returns

#### 3.3.2.3 but4press\_cbfunction()

```

void but4press_cbfunction (
    const struct device * dev,
    struct gpio_callback * cb,
    uint32_t pins )

```

If button 4 is pressed, Update Flag 4.

```

void but1press_cbfunction(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
{
    Flag_4 = 1;
}

```

#### Parameters

<i>arg3</i>	const struct device *dev, struct gpio_callback *cb, uint32_t pins.
-------------	--

#### Returns

No returns

#### 3.3.2.4 main()

```

void main (
    void )

```

Main funtion: Initialize semaphores and configure GPIO\_PIN (UART)

```

void main(void) {
    int err=0;
    printf("\n\r Illustration of the use of shmem + semaphores\n\r");
    int ret = 0;
    ret = gpio_pin_configure(gpio0_dev, BOARDBUT1, GPIO_INPUT | GPIO_PULL_UP);
    ret = gpio_pin_configure(gpio0_dev, BOARDBUT2, GPIO_INPUT | GPIO_PULL_UP);
    ret = gpio_pin_configure(gpio0_dev, BOARDBUT4, GPIO_INPUT | GPIO_PULL_UP);
    if (ret < 0) {

```

```

    printk("Error %d: Failed to configure BUT 1 \r", ret);
    return;
}
ret = gpio_pin_interrupt_configure(gpio0_dev, BOARDBUT1, GPIO_INT_EDGE_TO_ACTIVE);
ret = gpio_pin_interrupt_configure(gpio0_dev, BOARDBUT2, GPIO_INT_EDGE_TO_ACTIVE);
ret = gpio_pin_interrupt_configure(gpio0_dev, BOARDBUT4, GPIO_INT_EDGE_TO_ACTIVE);
if (ret != 0) {
    printk("Error %d: failed to configure interrupt on BUT1 pin \r", ret);
    return;
}
gpio_init_callback(&but1_cb_data, but1press_cbfunction, BIT(BOARDBUT1));
gpio_add_callback(gpio0_dev, &but1_cb_data);
gpio_init_callback(&but2_cb_data, but2press_cbfunction, BIT(BOARDBUT2));
gpio_add_callback(gpio0_dev, &but2_cb_data);
gpio_init_callback(&but4_cb_data, but4press_cbfunction, BIT(BOARDBUT4));
gpio_add_callback(gpio0_dev, &but4_cb_data);

err=0;
uint8_t welcome_mesg[] = "UART demo: Type a few chars in a row and then pause for a little while
    ...\n\r";
uint8_t rep_mesg[TXBUF_SIZE];

k_sem_init(&sem_a1a, 0, 1);
k_sem_init(&sem_ab, 0, 1);
k_sem_init(&sem_bc, 0, 1);
k_sem_init(&sem_bd, 0, 1);

    thread_A1_tid = k_thread_create(&thread_A1_data, thread_A1_stack,
        K_THREAD_STACK_SIZEOF(thread_A1_stack), thread_A1_code,
        NULL, NULL, NULL, thread_A1_prio, 0, K_NO_WAIT);
    thread_A_tid = k_thread_create(&thread_A_data, thread_A_stack,
        K_THREAD_STACK_SIZEOF(thread_A_stack), thread_A_code,
        NULL, NULL, NULL, thread_A_prio, 0, K_NO_WAIT);
    thread_B_tid = k_thread_create(&thread_B_data, thread_B_stack,
        K_THREAD_STACK_SIZEOF(thread_B_stack), thread_B_code,
        NULL, NULL, NULL, thread_B_prio, 0, K_NO_WAIT);
    thread_C_tid = k_thread_create(&thread_C_data, thread_C_stack,
        K_THREAD_STACK_SIZEOF(thread_C_stack), thread_C_code,
        NULL, NULL, NULL, thread_C_prio, 0, K_NO_WAIT);
    thread_D_tid = k_thread_create(&thread_D_data, thread_D_stack,
        K_THREAD_STACK_SIZEOF(thread_D_stack), thread_D_code,
        NULL, NULL, NULL, thread_D_prio, 0, K_NO_WAIT);

    return;
}

```

#### Parameters

<i>NO_args</i>	without arguments
----------------	-------------------

#### Returns

No returns

Welcome message

Create and init semaphores

Create tasks

### 3.3.2.5 thread\_A1\_code()

```

void thread_A1_code (
    void * argA,
    void * argB,
    void * argC )

```



É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por “ab” e no final faz give do semáforo AB.

```
void thread_A1_code(void *argA , void *argB, void *argC)
{
    int64_t fin_time=0, release_time=0;
    int err=0;
    printk("Thread A1 init (periodic)\n");

    release_time = k_uptime_get() + SAMP_PERIOD_MS;
    while(1) {

        if (flag_flag==0){
            printk("Modo manual\n");
            printk("Modo key 2 e 4\n");
            k_sem_give(&sem_bc);
        }

        else{
            printk("Modo automatico\n");

            k_sem_give(&sem_ala);
        }
        fin_time = k_uptime_get();
        if( fin_time < release_time) {
            k_msleep(release_time - fin_time);
            release_time += SAMP_PERIOD_MS;
        }
    }
}
```

#### Parameters

<i>arg3</i>	void *argA , void *argB, void *argC.
-------------	--------------------------------------

#### Returns

No returns

#### Thread code prototypes

##### 3.3.2.6 thread\_A\_code()

```
void thread_A_code (
    void * argA,
    void * argB,
    void * argC )
```

É o valor da ADC que é guardado numa variável global (shared memory between tasks A/B) no nosso Código denominada por “ab” e no final faz give do semáforo AB.

```
void thread_A_code(void *argA , void *argB, void *argC)
{
    int err=0;

    printk("Thread A init\n");

    adc_dev = device_get_binding(DT_LABEL(ADC_NID));
    if (!adc_dev) {
        printk("ADC device_get_binding() failed\n");
    }
    err = adc_channel_setup(adc_dev, &my_channel_cfg);
    if (err) {
        printk("adc_channel_setup() failed with error code %d\n", err);
    }

    NRF_SAADC->TASKS_CALIBRATEOFFSET = 1;
    while(1) {
```

```

    k_sem_take(&sem_a1a, K_FOREVER);

    err=adc_sample();
    if(err) {
        printk("adc_sample() failed with error code %d\n",err);
    }
    else {
        if(adc_sample_buffer[0] > 1023) {
            printk("adc reading out of range\n");
        }
        else {

            ab=adc_sample_buffer[0];
        }
    }
    printk("Thread A set ab value to: %d ",ab);

    k_sem_give(&sem_ab);

}
}

```

#### Parameters

<i>arg3</i>	void *argA , void *argB, void *argC.
-------------	--------------------------------------

#### Returns

No returns

### 3.3.2.7 thread\_B\_code()

```

void thread_B_code (
    void * argA,
    void * argB,
    void * argC )

```

é feito o take do semáforo AB é realizado uma média das últimas 10 amostras calculadas na thread A e é feito um filtro rejeitando todos os valores que estejam abaixo ou acima de 10% da média, sendo que este output é colocado numa variável global (shared memory between tasks B/C) no nosso Código denominada por “cb” e no final faz give do semáforo BC.

```

void thread_B_code(void *argA , void *argB, void *argC)
{
    int Array_dados[len_dados]={0};
    int k=0;
    printk("Thread B init (sporadic, waits on a semaphore by task A)\n");
    while(1) {
        int sumador=0,somador_2=0,media=0, media_filtered=0;
        int contador=0;

        k_sem_take(&sem_ab, K_FOREVER);

        printk("Task B read ab value: %d\n",ab);
        for(int k=len_dados-1; k>0;k--){

            Array_dados[k]= Array_dados[k-1];
        }
        Array_dados[0]= ab;

        for(int i = 0; i < len_dados; i++){
            if(Array_dados[i] != 0){
                sumador = sumador + Array_dados[i];
            }
        }
        media=sumador/len_dados;
        contador=0;
    }
}

```

```

    for(int j = 0; j < len_dados; j++){
        if(Array_dados[j] < (media - media*0.1) || Array_dados[j] > (media + media*0.1))
            somador_2=somador_2;
        else{
            somador_2 = somador_2 + Array_dados[j];
            contador =contador +1;
        }
    }
    if(somador_2 != 0)
        media_filtered=somador_2/contador;
    else
        media_filtered = 0;
    bd=ab;
    printk("Thread B set bc value to: %d\n",bc);
    k_sem_give(&sem_bd);
}
}

```

#### Parameters

<i>arg3</i>	void *argA , void *argB, void *argC.
-------------	--------------------------------------

#### Returns

No returns

array de dados da adc

#### 3.3.2.8 thread\_C\_code()

```

void thread_C_code (
    void * argA,
    void * argB,
    void * argC )
void thread_C_code(void *argA , void *argB, void *argC)
{
    const struct device *gpio0_dev;
    const struct device *pwm0_dev;
    int ret=0;
    unsigned int dcValue[]={100,90,80,70,60,50,40,30,20,10,0};
    unsigned int dcIndex=0;
    unsigned int pwmPeriod_us = 100;
    printk("Thread C init (sporadic, waits on a semaphore by task B)\n");

    gpio0_dev = device_get_binding(DT_LABEL(GPIO0_NID));
    if (gpio0_dev == NULL) {
        printk("Error: Failed to bind to GPIO0\nr");
        return;
    }

    pwm0_dev = device_get_binding(DT_LABEL(PWM0_NID));
    if (pwm0_dev == NULL) {
        printk("Error: Failed to bind to PWM0\n r");
        return;
    }

    while(1) {
        k_sem_take(&sem_bc, K_FOREVER);
        ret=0;

        if(Flag_2) {
            dcIndex++;
            if(dcIndex == 11)
                dcIndex = 0;
            Flag_2 = 0;
            printk("PWM DC value set to %u %%\n\r",dcValue[dcIndex]);
            ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
                pwmPeriod_us, (unsigned int)((pwmPeriod_us*dcValue[dcIndex])/100), PWM_POLARITY_NORMAL);
            if (ret) {
                printk("Error %d: failed to set pulse width\n", ret);
            }
            return;
        }
    }
}

```

```

    }
}
if(Flag_4) {
    if(dcIndex == 0)
        dcIndex = 11;
    dcIndex--;
    Flag_4 = 0;
    printk("PWM DC value set to %u %%\n\r",dcValue[dcIndex]);
    ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
        pwmPeriod_us,(unsigned int)((pwmPeriod_us*dcValue[dcIndex])/100), PWM_POLARITY_NORMAL);
    if (ret) {
        printk("Error %d: failed to set pulse width\n", ret);
    }
    return;
}

printk("Task C - PWM: %u % n", (unsigned int)((pwmPeriod_us*bc)/1023)/10));
}
}

```

#### Parameters

<i>arg3</i>	void *argA , void *argB, void *argC.
-------------	--------------------------------------

#### Returns

No returns

Prints dutty-cycle

### 3.3.2.9 thread\_D\_code()

```

void thread_D_code (
    void * argA,
    void * argB,
    void * argC )

```

Takes one adc\_sample.

```

void thread_D_code(void *argA , void *argB, void *argC)
{
    const struct device *gpio0_dev;
    const struct device *pwm0_dev;
    int ret=0;
    unsigned int dcValue[]={100,90,80,70,60,50,40,30,20,10,0};
    unsigned int dcIndex=0;
    unsigned int pwmPeriod_us = 100;
    printk("Thread C init (sporadic, waits on a semaphore by task B)\n");

    gpio0_dev = device_get_binding(DT_LABEL(GPIO0_NID));
    if (gpio0_dev == NULL) {
        printk("Error: Failed to bind to GPIO0\n\r");
        return;
    }

    pwm0_dev = device_get_binding(DT_LABEL(PWM0_NID));
    if (pwm0_dev == NULL) {
        printk("Error: Failed to bind to PWM0\n r");
        return;
    }

    while(1) {
        k_sem_take(&sem_bd, K_FOREVER);
        printk("Valor lido para automatico %d\n r",bd);
        ret=0;

        if(bd<500) {

            ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
                pwmPeriod_us,(unsigned int)(pwmPeriod_us), PWM_POLARITY_NORMAL);
            if (ret) {

```

```
        printk("Error %d: failed to set pulse width\n", ret);
    return;
    }
else {
    ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
        pwmPeriod_us, (unsigned int) (0), PWM_POLARITY_NORMAL);
    if (ret) {
        printk("Error %d: failed to set pulse width\n", ret);
    return;
    }
}
}
```

#### Parameters

<i>NO_args</i>	without arguments
<i>arg3</i>	void *argA , void *argB, void *argC.

```
static int adc_sample(void)
{
    int ret;
    const struct adc_sequence sequence = {
        .channels = BIT(ADC_CHANNEL_ID),
        .buffer = adc_sample_buffer,
        .buffer_size = sizeof(adc_sample_buffer),
        .resolution = ADC_RESOLUTION,
    };
    if (adc_dev == NULL) {
        printk("adc_sample(): error, must bind to adc first \r");
        return -1;
    }
    ret = adc_read(adc_dev, &sequence);
    if (ret) {
        printk("adc_read() failed with code %d\n", ret);
    }
    return ret;
}
```

#### Parameters

<i>NO_args</i>	without arguments
----------------	-------------------

#### Returns

Read ADC\_sample value (static int)



# Index

a1a  
    main.c, 19

ab  
    main.c, 19

ADC\_ACQUISITION\_TIME  
    main.c, 7

ADC\_CHANNEL\_ID  
    main.c, 7

ADC\_CHANNEL\_INPUT  
    main.c, 8

adc\_dev  
    main.c, 19

ADC\_GAIN  
    main.c, 8

ADC\_NID  
    main.c, 8

ADC\_REFERENCE  
    main.c, 8

ADC\_RESOLUTION  
    main.c, 8

bc  
    main.c, 19

bd  
    main.c, 19

BOARDBUT1  
    main.c, 8

BOARDBUT2  
    main.c, 8, 9

BOARDBUT4  
    main.c, 9

BOARDLED1  
    main.c, 9

BUFFER\_SIZE  
    main.c, 9

but1press\_cbfunction  
    main.c, 11  
    main.h, 24

but2press\_cbfunction  
    main.c, 12  
    main.h, 24

but4press\_cbfunction  
    main.c, 12  
    main.h, 25

cmake\_minimum\_required  
    CMakeLists.txt, 5

CMakeLists.txt, 5  
    cmake\_minimum\_required, 5

FATAL\_ERR  
    main.c, 9

Flag\_1  
    main.c, 19

Flag\_2  
    main.c, 20

Flag\_3  
    main.c, 20

Flag\_4  
    main.c, 20

flag\_flag  
    main.c, 20

gpio0\_dev  
    main.c, 20

GPIO0\_NID  
    main.c, 9

K\_THREAD\_STACK\_DEFINE  
    main.c, 13

len\_dados  
    main.c, 9

main  
    main.c, 13  
    main.h, 25

main.c, 5  
    a1a, 19  
    ab, 19  
    ADC\_ACQUISITION\_TIME, 7  
    ADC\_CHANNEL\_ID, 7  
    ADC\_CHANNEL\_INPUT, 8  
    adc\_dev, 19  
    ADC\_GAIN, 8  
    ADC\_NID, 8  
    ADC\_REFERENCE, 8  
    ADC\_RESOLUTION, 8  
    bc, 19  
    bd, 19  
    BOARDBUT1, 8  
    BOARDBUT2, 8, 9  
    BOARDBUT4, 9  
    BOARDLED1, 9  
    BUFFER\_SIZE, 9  
    but1press\_cbfunction, 11  
    but2press\_cbfunction, 12  
    but4press\_cbfunction, 12  
    FATAL\_ERR, 9  
    Flag\_1, 19

Flag\_2, [20](#)  
 Flag\_3, [20](#)  
 Flag\_4, [20](#)  
 flag\_flag, [20](#)  
 gpio0\_dev, [20](#)  
 GPIO0\_NID, [9](#)  
 K\_THREAD\_STACK\_DEFINE, [13](#)  
 len\_dados, [9](#)  
 main, [13](#)  
 MAIN\_SLEEP\_TIME\_MS, [9](#)  
 my\_timer, [20](#)  
 PWM0\_NID, [10](#)  
 rep\_mesg, [20](#)  
 RX\_TIMEOUT, [10](#)  
 RXBUF\_SIZE, [10](#)  
 SAMP\_PERIOD\_MS, [10](#)  
 sem\_a1a, [20](#)  
 sem\_ab, [21](#)  
 sem\_bc, [21](#)  
 sem\_bd, [21](#)  
 STACK\_SIZE, [10](#)  
 thread\_A1\_code, [14](#)  
 thread\_A1\_data, [21](#)  
 thread\_A1\_prio, [10](#)  
 thread\_A1\_tid, [21](#)  
 thread\_A\_code, [15](#)  
 thread\_A\_data, [21](#)  
 thread\_A\_prio, [10](#)  
 thread\_A\_tid, [21](#)  
 thread\_B\_code, [16](#)  
 thread\_B\_data, [21](#)  
 thread\_B\_prio, [11](#)  
 thread\_B\_tid, [22](#)  
 thread\_C\_code, [17](#)  
 thread\_C\_data, [22](#)  
 thread\_C\_prio, [11](#)  
 thread\_C\_tid, [22](#)  
 thread\_D\_code, [17](#)  
 thread\_D\_data, [22](#)  
 thread\_D\_prio, [11](#)  
 thread\_D\_tid, [22](#)  
 TXBUF\_SIZE, [11](#)  
 uart\_cfg, [22](#)  
 uart\_dev, [22](#)  
 UART\_NID, [11](#)  
 uart\_rx\_rdy\_flag, [23](#)  
 welcome\_mesg, [23](#)  
 main.h, [23](#)  
   but1press\_cbfunction, [24](#)  
   but2press\_cbfunction, [24](#)  
   but4press\_cbfunction, [25](#)  
   main, [25](#)  
   thread\_A1\_code, [26](#)  
   thread\_A\_code, [27](#)  
   thread\_B\_code, [28](#)  
   thread\_C\_code, [29](#)  
   thread\_D\_code, [30](#)  
 MAIN\_SLEEP\_TIME\_MS  
   main.c, [9](#)  
 my\_timer  
   main.c, [20](#)  
 PWM0\_NID  
   main.c, [10](#)  
 rep\_mesg  
   main.c, [20](#)  
 RX\_TIMEOUT  
   main.c, [10](#)  
 RXBUF\_SIZE  
   main.c, [10](#)  
 SAMP\_PERIOD\_MS  
   main.c, [10](#)  
 sem\_a1a  
   main.c, [20](#)  
 sem\_ab  
   main.c, [21](#)  
 sem\_bc  
   main.c, [21](#)  
 sem\_bd  
   main.c, [21](#)  
 STACK\_SIZE  
   main.c, [10](#)  
 thread\_A1\_code  
   main.c, [14](#)  
   main.h, [26](#)  
 thread\_A1\_data  
   main.c, [21](#)  
 thread\_A1\_prio  
   main.c, [10](#)  
 thread\_A1\_tid  
   main.c, [21](#)  
 thread\_A\_code  
   main.c, [15](#)  
   main.h, [27](#)  
 thread\_A\_data  
   main.c, [21](#)  
 thread\_A\_prio  
   main.c, [10](#)  
 thread\_A\_tid  
   main.c, [21](#)  
 thread\_B\_code  
   main.c, [16](#)  
   main.h, [28](#)  
 thread\_B\_data  
   main.c, [21](#)  
 thread\_B\_prio  
   main.c, [11](#)  
 thread\_B\_tid  
   main.c, [22](#)  
 thread\_C\_code  
   main.c, [17](#)  
   main.h, [29](#)  
 thread\_C\_data  
   main.c, [22](#)



thread\_C\_prio  
    main.c, [11](#)  
thread\_C\_tid  
    main.c, [22](#)  
thread\_D\_code  
    main.c, [17](#)  
    main.h, [30](#)  
thread\_D\_data  
    main.c, [22](#)  
thread\_D\_prio  
    main.c, [11](#)  
thread\_D\_tid  
    main.c, [22](#)  
TXBUF\_SIZE  
    main.c, [11](#)  
  
uart\_cfg  
    main.c, [22](#)  
uart\_dev  
    main.c, [22](#)  
UART\_NID  
    main.c, [11](#)  
uart\_rx\_rdy\_flag  
    main.c, [23](#)  
  
welcome\_mesg  
    main.c, [23](#)