Assigment 4 - Implementing cooperative tasks in Zephyr

V1.1

Generated by Doxygen 1.8.17

Chapter 1

Bug List

File main.c

No known bugs.

File semaphore.h

No known bugs.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

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()

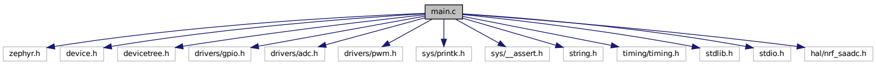
```
cmake_minimum_required (
            VERSION 3.20.   0 )
```

3.2 main.c File Reference

main.c It reads the input voltage from an analog sensor, digitally filters the signal and outputs it.

```
#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 <hal/nrf_saadc.h>
```

Include dependency graph for main.c:



Macros

- #define len_dados 10
- #define STACK_SIZE 1024
- #define thread_A_prio 1
- #define thread_B_prio 1
- #define thread_C_prio 1
- #define thread_A_period 1000 /∗∗ Set to have the same period as the PWM, 1ms∗/
- #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

Functions

- 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)
- void thread_A_code (void ∗argA, void ∗argB, void ∗argC)

  ê o valor da ADCe guarda 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)

  é feito o take do semáforo BC e é criado um pwm signal que é depois aplicado a um led. Todo este processo é repetido período após período.
- void main (void)

  Main funtion: Initialize semaphores.

Variables

- struct k_thread thread_A_data
- struct k_thread thread_B_data
- struct k_thread thread_C_data
- k_tid_t thread_A_tid
- k_tid_t thread_B_tid
- k_tid_t thread_C_tid
- int ab = 0
- int bc = 0
- struct k_sem sem_ab
- struct k_sem sem_bc
- struct k_timer my_timer
- const struct device ∗ adc_dev = NULL

3.2.1 Detailed Description

main.c It reads the input voltage from an analog sensor, digitally filters the signal and outputs it.

Author

Ana Sousa, Frederico Moreira, Pedro Rodrigues

Date

31 March 2022

Bug No known bugs.

3.2.2 Macro Definition Documentation

3.2.2.1 ADC_ACQUISITION_TIME

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

3.2.2.2 ADC_CHANNEL_ID

```
#define ADC_CHANNEL_ID 1
```

3.2.2.3 ADC_CHANNEL_INPUT

```
#define ADC_CHANNEL_INPUT NRF_SAADC_INPUT_AIN1
```

3.2.2.4 ADC_GAIN

```
#define ADC_GAIN ADC_GAIN_1_4
```

3.2.2.5 ADC_NID

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

ADC definitions and includes

3.2.2.6 ADC_REFERENCE

```
#define ADC_REFERENCE ADC_REF_VDD_1_4
```

3.2.2.7 ADC_RESOLUTION

```
#define ADC_RESOLUTION 10
```

3.2.2.8 BOARDLED1

```
#define BOARDLED1 0x0d
```

3.2.2.9 BUFFER_SIZE

```
#define BUFFER_SIZE 1
```

3.2.2.10 GPIO0_NID

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

Refer to dts file

3.2.2.11 len_dados

```
#define len_dados 10
```

Number of samples for the average

3.2.2.12 PWM0_NID

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

3.2.2.13 STACK_SIZE

```
#define STACK_SIZE 1024
```

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

3.2.2.14 thread_A_period

```
#define thread_A_period 1000 /** Set to have the same period as the PWM, 1ms*/
```

Therad periodicity (in ms)

3.2.2.15 thread_A_prio

```
#define thread_A_prio 1
```

Thread scheduling priority

3.2.2.16 thread_B_prio

```
#define thread_B_prio 1
```

3.2.2.17 thread_C_prio

```
#define thread_C_prio 1
```

3.2.3 Function Documentation

3.2.3.1 K_THREAD_STACK_DEFINE() [1/3]

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

Create thread stack space

3.2.3.2 K_THREAD_STACK_DEFINE() [2/3]

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

3.2.3.3 K_THREAD_STACK_DEFINE() [3/3]

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

3.2.3.4 main()

```
void main (
             void  )
```

Main funtion: Initialize semaphores.

Main function

3.2.3.5 thread_A_code()

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

ê o valor da ADCe guarda 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

3.2.3.6 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);
        Array_dados[0]= ab;
        Array_dados[(k+1)%10]= Array_dados[(k)%10];
        k=k+1;


       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;
        bc=media;
        printk("Thread B set bc value to: %d\n",bc);
        k_sem_give(&sem_bc);
  }
}
```

Parameters

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

Returns

No returns

3.2.3.7 thread_C_code()

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

é feito o take do semáforo BC e é criado um pwm signal que é depois aplicado a um led. Todo este processo é repetido período após período.

```
 *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 pwmPeriod_us = 1000;
    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;

        ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
            pwmPeriod_us,(unsigned int)((pwmPeriod_us*bc)/1023), 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

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

Returns

No returns

3.2.4 Variable Documentation

3.2.4.1 ab

```
int ab = 0
```

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

3.2.4.2 adc_dev

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

3.2.4.3 bc

```
int bc = 0
```

3.2.4.4 my_timer

```
struct k_timer my_timer
```

Global vars

3.2.4.5 sem_ab

```
struct k_sem sem_ab
```

Semaphores for task synch

3.2.4.6 sem_bc

```
struct k_sem sem_bc
```

3.2.4.7 thread_A_data

```
struct k_thread thread_A_data
```

Create variables for thread data

3.2.4.8 thread_A_tid

```
k_tid_t thread_A_tid
```

Create task IDs

3.2.4.9 thread_B_data

```
struct k_thread thread_B_data
```

3.2.4.10 thread_B_tid

```
k_tid_t thread_B_tid
```

3.2.4.11 thread_C_data

```
struct k_thread thread_C_data
```

3.2.4.12 thread_C_tid

```
k_tid_t thread_C_tid
```

3.3 semaphore.h File Reference

The system to implement does a basic processing of an analog signal. It reads the input voltage from an analog sensor, digitally filters the signal and outputs it using a semaphore.

Functions

- void main (void)

  Main funtion: Initialize semaphores.
- void thread_A_code (void ∗argA, void ∗argB, void ∗argC)

  ê o valor da ADCe guarda 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)

  é feito o take do semáforo BC e é criado um pwm signal que é depois aplicado a um led. Todo este processo é repetido período após período.

3.3.1 Detailed Description

The system to implement does a basic processing of an analog signal. It reads the input voltage from an analog sensor, digitally filters the signal and outputs it using a semaphore.

Contains the functions needed to process the analog signal

Author

Frederico Moreira, Ana Sousa, Pedro Rodrigues

Date

31 May 2022

Bug No known bugs.

3.3.2 Function Documentation

3.3.2.1 main()

```
void main (
               void   )
```

Main funtion: Initialize semaphores.

```
   void main(void) {
    printf("\n\r Illustration of the use of shmem + semaphores\n\r");

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


    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_B_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);

    return;
}
```

Parameters

NO_args	without arguments

Returns

No returns

Main function

3.3.2.2 thread_A_code()

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

ê o valor da ADCe guarda 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)
{

    int64_t fin_time=0, release_time=0;
    int err=0;

    printk("Thread A init (periodic)\n");

    release_time = k_uptime_get() + thread_A_period;
    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) {

        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 \n",ab);

        k_sem_give(&sem_ab);


        fin_time = k_uptime_get();
        if( fin_time < release_time) {
            k_msleep(release_time - fin_time);
            release_time += thread_A_period;
        }
    }
}
```

Parameters

arg3	void ∗argA , void ∗argB, void ∗argC.

Returns

No returns

Thread code prototypes

3.3.2.3 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);
        Array_dados[0]= ab;
        Array_dados[(k+1)%10]= Array_dados[(k)%10];
        k=k+1;


       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;
        bc=media;
        printk("Thread B set bc value to: %d\n",bc);
        k_sem_give(&sem_bc);
  }
}
```

Parameters

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

Returns

No returns

3.3.2.4 thread_C_code()

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

é feito o take do semáforo BC e é criado um pwm signal que é depois aplicado a um led. Todo este processo é repetido período após período.

```
 *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 pwmPeriod_us = 1000;
    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;

        ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
              pwmPeriod_us,(unsigned int)((pwmPeriod_us*bc)/1023), 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

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

Returns

No returns

Index