Assigment 4 - Implementing cooperative tasks in Zephyr

V1.1

Generated by Doxygen 1.8.17

Chapter 1

Bug List

File fifo.h

No known bugs.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 data_item_t Struct Reference

Data Fields

- void * fifo_reserved
- uint16_t data

4.1.1 Field Documentation

4.1.1.1 data

```
uint16_t data_item_t::data
```

4.1.1.2 fifo_reserved

```
void* data_item_t::fifo_reserved
```

The documentation for this struct was generated from the following file:

- main.c

Chapter 5

File Documentation

5.1 CMakeLists.txt File Reference

Functions

- cmake_minimum_required (VERSION 3.20.0) find_package(Zephyr REQUIRED HINTS $ENV

5.1.1 Function Documentation

5.1.1.1 cmake_minimum_required()

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

5.2 fifo.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 fifo.

Functions

- void main (void)

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

  Read the adc value and save it.
- void thread_B_code (void ∗argA, void ∗argB, void ∗argC)

  calculates the average of 10 values read from the adc and if the value is outside 10% it is rejected.
- void thread_C_code (void ∗argA, void ∗argB, void ∗argC)

  Sets the PWM DC value to the average of the samples got from ADC module in thread B.

5.2.1 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 fifo.

Contains the functions needed to process the analog signal

Author

Frederico Moreira, Ana Sousa, Pedro Rodrigues

Date

31 May 2022

Bug No known bugs.

5.2.2 Function Documentation

5.2.2.1 main()

```
void main (
              void  )
```

Main funtion: Initialize semaphores.

```
    printk("\n\r IPC via FIFO example \n\r");


    k_fifo_init(&fifo_ab);
    k_fifo_init(&fifo_bc);


    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

5.2.2.2 thread_A_code()

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

Read the adc value and save it.

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

    int64_t fin_time=0, release_time=0;
    long int nact = 0;
    int err=0;
    struct data_item_t data_ab;

    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);
    }


    while(1) {


        printk("\n\nThread A instance %ld released at time: %lld (ms). \n",++nact, k_uptime_get());

        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 {

                data_ab.data = adc_sample_buffer[0];

            }
        }
        k_fifo_put(&fifo_ab, &data_ab);
        printk("Thread A data in fifo_ab: %d\n",data_ab.data);

        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

5.2.2.3 thread_B_code()

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

calculates the average of 10 values read from the adc and if the value is outside 10% it is rejected.

```
 *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

5.2.2.4 thread_C_code()

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

Sets the PWM DC value to the average of the samples got from ADC module in thread B.

```
 *void thread_C_code(void *argA , void *argB, void *argC)
{
    long int nact = 0;
    struct data_item_t *data_bc;
    printk("Thread C init (sporadic, waits on a semaphore by task A)\n");
    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) {
        data_bc = k_fifo_get(&fifo_bc, K_FOREVER);
        printk("Thread C instance %5ld released at time: %lld (ms). \n",++nact, k_uptime_get());
        printk("Task C read bc value: %d\n",data_bc->data);
        ret=0;
        ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
              pwmPeriod_us,(unsigned int)((pwmPeriod_us*data_bc->data)/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*data_bc->data)/1023)/10));
  }
}
```

Parameters

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

Returns

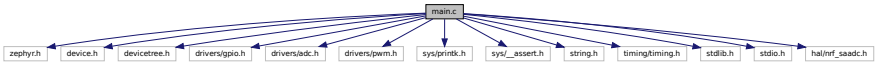
No returns

5.3 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 <hal/nrf_saadc.h>
```

Include dependency graph for main.c:



Data Structures

- struct data_item_t

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
- #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 /∗∗ Analog 1 - Port P0.03 ∗/
- #define BUFFER_SIZE 1
- #define GPIO0_NID DT_NODELABEL(gpio0)
- #define PWM0_NID DT_NODELABEL(pwm0)
- #define BOARDLED1 0x0d /∗∗ LED 1 ∗/

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 ∗, void ∗, void ∗)

  Read the adc value and save it.
- void thread_B_code (void ∗, void ∗, void ∗)

  calculates the average of 10 values read from the adc and if the value is outside 10% it is rejected.
- void thread_C_code (void ∗, void ∗, void ∗)

  Sets the PWM DC value to the average of the samples got from ADC module in thread B.
- 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
- struct k_fifo fifo_ab
- struct k_fifo fifo_bc
- struct k_timer my_timer
- const struct device * adc_dev = NULL

5.3.1 Macro Definition Documentation

5.3.1.1 ADC_ACQUISITION_TIME

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

5.3.1.2 ADC_CHANNEL_ID

```
#define ADC_CHANNEL_ID 1
```

5.3.1.3 ADC_CHANNEL_INPUT

```
#define ADC_CHANNEL_INPUT NRF_SAADC_INPUT_AIN1 /** Analog 1 - Port P0.03 */
```

5.3.1.4 ADC_GAIN

```
#define ADC_GAIN ADC_GAIN_1_4
```

5.3.1.5 ADC_NID

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

ADC definitions and includes

5.3.1.6 ADC_REFERENCE

```
#define ADC_REFERENCE ADC_REF_VDD_1_4
```

5.3.1.7 ADC_RESOLUTION

```
#define ADC_RESOLUTION 10
```

5.3.1.8 BOARDLED1

```
#define BOARDLED1 0x0d /** LED 1 */
```

5.3.1.9 BUFFER_SIZE

```
#define BUFFER_SIZE 1
```

5.3.1.10 GPIO0_NID

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

Refer to dts file

5.3.1.11 len_dados

```
#define len_dados 10
```

Number of samples for the average

5.3.1.12 PWM0_NID

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

5.3.1.13 STACK_SIZE

```
#define STACK_SIZE 1024
```

5.3.1.14 thread_A_period

```
#define thread_A_period 1000
```

5.3.1.15 thread_A_prio

```
#define thread_A_prio 1
```

5.3.1.16 thread_B_prio

```
#define thread_B_prio 1
```

5.3.1.17 thread_C_prio

```
#define thread_C_prio 1
```

5.3.2 Function Documentation

5.3.2.1 K_THREAD_STACK_DEFINE() [1/3]

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

5.3.2.2 K_THREAD_STACK_DEFINE() [2/3]

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

5.3.2.3 K_THREAD_STACK_DEFINE() [3/3]

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

5.3.2.4 main()

```
void main (
              void  )
```

Main funtion: Initialize semaphores.

```
    printk("\n\r IPC via FIFO example \n\r");


    k_fifo_init(&fifo_ab);
    k_fifo_init(&fifo_bc);


    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

5.3.2.5 thread_A_code()

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

Read the adc value and save it.

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

    int64_t fin_time=0, release_time=0;
    long int nact = 0;
    int err=0;
    struct data_item_t data_ab;

    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);
    }


    while(1) {


        printk("\n\nThread A instance %ld released at time: %lld (ms). \n",++nact, k_uptime_get());

        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 {

                data_ab.data = adc_sample_buffer[0];

            }
        }
        k_fifo_put(&fifo_ab, &data_ab);
        printk("Thread A data in fifo_ab: %d\n",data_ab.data);

        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

5.3.2.6 thread_B_code()

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

calculates the average of 10 values read from the adc and if the value is outside 10% it is rejected.

```
 *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

5.3.2.7 thread_C_code()

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

Sets the PWM DC value to the average of the samples got from ADC module in thread B.

```
 *void thread_C_code(void *argA , void *argB, void *argC)
{
    long int nact = 0;
    struct data_item_t *data_bc;
    printk("Thread C init (sporadic, waits on a semaphore by task A)\n");
    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) {
        data_bc = k_fifo_get(&fifo_bc, K_FOREVER);
        printk("Thread C instance %5ld released at time: %lld (ms). \n",++nact, k_uptime_get());
        printk("Task C read bc value: %d\n",data_bc->data);
        ret=0;
        ret = pwm_pin_set_usec(pwm0_dev, BOARDLED1,
              pwmPeriod_us,(unsigned int)((pwmPeriod_us*data_bc->data)/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*data_bc->data)/1023)/10));
  }
}
```

Parameters

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

Returns

No returns

5.3.3 Variable Documentation

5.3.3.1 adc_dev

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

5.3.3.2 fifo_ab

```
struct k_fifo fifo_ab
```

5.3.3.3 fifo_bc

```
struct k_fifo fifo_bc
```

5.3.3.4 my_timer

```
struct k_timer my_timer
```

Global vars

5.3.3.5 thread_A_data

```
struct k_thread thread_A_data
```

5.3.3.6 thread_A_tid

```
k_tid_t thread_A_tid
```

5.3.3.7 thread_B_data

```
struct k_thread thread_B_data
```

5.3.3.8 thread_B_tid

```
k_tid_t thread_B_tid
```

5.3.3.9 thread_C_data

```
struct k_thread thread_C_data
```

5.3.3.10 thread_C_tid

```
k_tid_t thread_C_tid
```

Index