Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

2021/2022

Architecture des ordinateurs

Rapport 02

Travail pratique 2 : Bouton-poussoir, assertions, horloge

Ecole	Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR)
Branche	Architecture des ordinateurs
Etudiants	Barras Simon, Terreaux Nicolas
Groupe	B01
Classe	ISC-IL-2d
Professeur	Supcik Jacques
GitLab	https://gitlab.forge.hefr.ch/ado/2021-2022/classe-supcik/groupe-B-01/tp01
Version/Date	V1 du 18.10.2021

1 Introduction

Le but de ce TP est de se familiariser avec différentes entrées/sorties, les assertions, les timers du langage C++.

Le TP se comporte **3 exercices**. **Le premier** qui consiste à allumer/éteindre la LED bleue à chaque pression du bouton central du Joystick. **Le deuxième** doit aussi allumer/éteindre la LED bleue en fonction du bouton mais cette fois en utilisant le polling périodique et en plus, il faut faire clignoter la LED verte pendant l'exécution du programme. **Le troisième** consiste à écrire un programme qui utilise le timer pour calculer le nombre d'itérations à faire pour obtenir une seconde.

2 Résumés

Ci-dessous, les différentes notions théoriques abordées lors ce TP. Elles sont catégorisées par niveau de compréhension.

2.1 Non Acquis

Nous ne pensons pas avoir de notion non-acquise.

2.2 A exercer

- CI/CD : Nous sommes désormais plus familiers avec CI/CD malgré encore des incertitudes.
- Polling périodique : Nous avons bien compris le concept mais nous ne sommes pas encore des professionnels.
- Asserts : Nous avons compris le fonctionnement des asserts mais nous avons encore de la peine à savoir quand et surtout où les utiliser.

2.3 Parfaitement acquis

- Boutons-poussoirs : Nous avons bien compris comment fonctionne un bouton. Les explications sur « PULL-UP/PULL-DOWN » sont très claires et nous avions déjà abordé ce sujet lors du cours technique numérique 1 en première année.
- Fonctionnement des timers et leurs utilités.

3 Points importants

Ce chapitre résume les points importants que nous avons fait durant ce TP.

3.1 « PlaterformIO »

Nous pouvons modifier le fichier de configuration Platform.io afin de configurer les actions réaliser lors de la compilation ou de l'upload.

Par exemple, nous pouvons définir des « Virtaual environnement » pour qu'il compile seulement les bons fichiers :

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
```

```
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[platformio]
default_envs = ex3

[env]
platform = ststm32
platform_packages =
  toolchain-gccarmnoneeabi@>=1.90000.0

board = disco_f412zg
framework = libopencm3
build_flags = -Wl,-u,_printf_float,-u,_scanf_float
test_transport = custom

monitor_speed = 115200



; By default, the check tool uses only "cppcheck".
; Uncomment the following line to also use "clang-tidy"

check_tool = cppcheck, clangtidy

[env:ex1]
src_filter = +<*> -<ex*.cpp> +<ex1.cpp>

[env:ex2]
src_filter = +<*> -<ex*.cpp> +<ex2.cpp>

[env:ex3]
src_filter = +<*> -<ex*.cpp> +<ex3.cpp>
```

Cette configuration compilera tous les codes sauf les fichiers « ex1.cpp » et « ex2.cpp ». Pour modifier ce comportement il suffit de changer la ligne « default_envs » avec le nom de l'environnement que nous voulons utiliser.

3.2 Polling périodique

Le polling périodique est une technique qui consiste à demander régulièrement l'état du bouton et de réagir en cas de changement. Par exemple, dans l'exercice 1, nous interrogeons le bouton en continu ce qui pouvait entrainer de problèmes.

3.3 Timer

Le microcontrôleur possède plusieurs timers. De base, ils sont éteints pour économiser de l'énergie et nous devons donc les activer. Ces timers appellent la méthode « timX_isr() » d'après la fréquence que nous leur avons donné.

4 Questions

Durant ce TP, nous devions répondre à deux questions. Les voici :

1) La variable `system_millis` est de type `uin32_t`. Dans combien de temps aura-t-elle atteint sa valeur maximale ? Donnez la réponse en année(s), mois(s) semaine(s) jour(s) minute(s) seconde(s)

Pour répondre à cette question, nous avons converti les 32 bits en ms :

2^32 – 1 = **4'294'967'295 ms**

Nous nous sommes ensuite rendus sur un site qui converti les ms en années, mois, minutes, secondes :

Figure 1: Screenshot du résultat sur le site : https://www.unitjuggler.com/convertir-time-de-s-en-yr-365.html

Donc elle aura atteint sa valeure maximale dans **49 jours, 17h, 2min, 47sec et 295ms**.

2) Quand la variable `system_millis` dépassera sa valeur maximale, est-ce que la procédure `SystickDelayMilliseconds` fonctionne encore correctement ?

La procédure ne fonctionnera plus correctement. En effet, lorsqu'on dépasse 2^32, le compilateur génèrera une erreur de type : `System.OverflowException`.

5 Exercices

Voici les différents codes de notre travail pratique.

5.1 Ex1

```
// Copyright 2021 Haute école d'ingénierie et d'architecture de Fribourg
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
// See the License for the specific language governing permissions and
// limitations under the License.

/*****************************************************************************
 * @file ex1.cpp
 * @author Simon Barras <simon.barras@edu.hefr.ch>, Nicolas Terreaux
<nicolas.terreaux@edu.hefr.ch>
 *
 * @brief Turn on/off blue led with a press button
 *
 * @date 2021-10-12
 * @version 0.1.0
 *****************************************************************************/

#include <DiscoConsole.h>
#include <libopencm3/cm3/assert.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/rcc.h>
#include <stdint.h>
#include <stdio.h>
#include <DiscoAssert.hpp>

constexpr auto kRccLedPort    = RCC_GPIOE;
constexpr auto kRccButtonPort = RCC_GPIOA;
constexpr auto kLedPort       = GPIOE;
constexpr auto kLedPin        = GPIO3;
constexpr auto kButtonPort    = GPIOA;
constexpr auto kButtonPin     = GPIO0;


static void setup(void)
{
    rcc_clock_setup_pll(&rcc_hse_8mhz_3v3[RCC_CLOCK_3V3_84MHZ]);
    rcc_periph_clock_enable(kRccLedPort);
    rcc_periph_clock_enable(kRccButtonPort);
    gpio_mode_setup(kLedPort, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, kLedPin);
    gpio_mode_setup(kButtonPort, GPIO_MODE_INPUT, GPIO_PUPD_PULLDOWN,
kButtonPin);
}

//Listen the button and when it's clicked, blue LED turn on/off
static void toggleOnClick()
{
    uint16_t oldRes = 0;
    uint16_t res    = 0;
    while (true) {
        res = gpio_get(kButtonPort, kButtonPin);
        if (res != 0) {
            //Check if has been "reset"
```

```
            if (res != oldRes) {
                //printf("Res: %i\t oldRes: %i\n", res, oldRes);
                gpio_toggle(kLedPort, kLedPin);
            }
        }
        oldRes = res;
    }
}

int main()
{
    setup();
    DiscoConsoleSetup();
    toggleOnClick();
    return 0;
}
```

5.2 Ex2

```
// Copyright 2021 Haute école d'ingénierie et d'architecture de Fribourg
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/*************************
 * @file ex2.cpp
 * @author Simon Barras <simon.barras@edu.hefr.ch>, Nicolas Terreaux
<nicolas.terreaux@edu.hefr.ch>
 *
 * @brief Turn on/off blue led with a press button and blinks green LED
 *
 * @date 2021-10-16
 * @version 0.1.0
 ************************/

#include <libopencm3/cm3/nvic.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/rcc.h>
```

```
#include <libopencm3/stm32/timer.h>
#include <Systick.hpp>

constexpr auto kPollingFrequency = 100;  // in Hz
constexpr auto kRccBtnPort       = RCC_GPIOA;
constexpr auto kRccLedPort       = RCC_GPIOE;
constexpr auto kLedPort          = GPIOE;
constexpr auto kLedBluePin       = GPIO3;
constexpr auto kButtonPort       = GPIOA;
constexpr auto kButtonPin        = GPIO0;
constexpr auto kLedGreenPin      = GPIO0;
constexpr auto kSleepLoop        = 7000000;

static void setup(void)
{
    // activate the clock for the GPIOs
    rcc_periph_clock_enable(RCC_TIM2);
    rcc_periph_reset_pulse(RST_TIM2);

    // timer configuration
    timer_set_period(TIM2, rcc_apb2_frequency / kPollingFrequency);
    timer_enable_irq(TIM2, TIM_DIER_UIE);
    timer_enable_counter(TIM2);

    // controller interruption
    nvic_enable_irq(NVIC_TIM2_IRQ);

    rcc_clock_setup_pll(&rcc_hse_8mhz_3v3[RCC_CLOCK_3V3_84MHZ]);
    rcc_periph_clock_enable(kRccBtnPort);
    rcc_periph_clock_enable(kRccLedPort);
    gpio_mode_setup(kLedPort, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, kLedGreenPin);
    gpio_mode_setup(kLedPort, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, kLedBluePin);
    gpio_mode_setup(kButtonPort, GPIO_MODE_INPUT, GPIO_PUPD_PULLDOWN,
kButtonPin);
}

void tim2_isr(void)
{
    static int prevBtn = 0;
    timer_clear_flag(TIM2, TIM_SR_UIF);  // acknowledge interrupt

    uint16_t btn = gpio_get(kButtonPort, kButtonPin);
    if ((btn != prevBtn) && (btn != 0)) {
        gpio_toggle(kLedPort, kLedBluePin);
    }
    prevBtn = btn;
}

int main()
```

```
{
    setup();
    SystickSetup();

    while (1) {
        // Blinks the green LED
        gpio_toggle(kLedPort, kLedGreenPin);
        SystickDelayMilliseconds(200);
    }
    return 0;
}
```

5.3 Ex3

```
// Copyright 2021 Haute école d'ingénierie et d'architecture de Fribourg
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/*****************************************************************************
 * @file ex3.cpp
 * @author Simon Barras <simon.barras@edu.hefr.ch>, Nicolas Terreaux
<nicolas.terreaux@edu.hefr.ch>
 *
 * @brief Find the number of instruction to run 1 sec
 *
 * @date 2021-10-17
 * @version 0.1.0
 *****************************************************************************/

#include <DiscoConsole.h>
#include <libopencm3/cm3/assert.h>
#include <libopencm3/cm3/nvic.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/rcc.h>
#include <libopencm3/stm32/timer.h>
#include <stdint.h>
#include <stdio.h>
```

```
#include <DiscoAssert.hpp>
#include <Systick.hpp>

constexpr auto kPollingFrequency         = 1000;  // in Hz
constexpr uint64_t kNbInstructionDefault = 500;
constexpr auto kOneSec                   = 1000;
static volatile uint32_t system_millis   = 0;  // current time

static void setup(void)
{
    rcc_periph_clock_enable(RCC_TIM2);
    rcc_periph_reset_pulse(RST_TIM2);

    // timer configuration
    timer_set_period(TIM2, rcc_apb2_frequency / kPollingFrequency);
    timer_enable_irq(TIM2, TIM_DIER_UIE);
    timer_enable_counter(TIM2);

    nvic_enable_irq(NVIC_TIM2_IRQ);
}

void tim2_isr(void)
{
    timer_clear_flag(TIM2, TIM_SR_UIF);  // acknowledge interrupt
    system_millis++;
}

int main()
{
    DiscoConsoleSetup();
    SystickSetup();
    setup();
    tim2_isr();

    auto time          = 0;
    auto nbIstructions = kNbInstructionDefault;

    printf("start\n");

    while (time != kOneSec) {
        auto timeStart = system_millis;
        cm3_assert(nbIstructions >= 0);
        for (uint64_t i = 0; i < nbIstructions; i++) {
            // Prevents the compiler's optimization pass from removing
this while loop
            asm volatile("nop");
        }
        auto timeStop = system_millis;
        time          = timeStop - timeStart;
```

```
        if (time == 0) {
            time = 1;
        }
        cm3_assert(time > 0);
        float quotient = (float) kOneSec / (float) time;
        printf("%d millis with %lu instructions -> quotient %f\n", time,
nbIstructions, quotient);
        nbIstructions = nbIstructions * quotient;
    }
    return 0;
```

Et voici le résultat que nous avons eu :

```
start
1 millis with 500 instructions -> quotient 1000.000000
282 millis with 500000 instructions -> quotient 3.546099
1001 millis with 1773049 instructions -> quotient 0.999001
1001 millis999 millis with 1769507 instructions -> quotient 1.001001
1000 millis with 1771278 instructions -> quotient 1.000000
```

Nous avons remarqué que nous n'avions pas toujours le même résultat ce qui est normal.

6 Conclusion

Nous avons apprécié effectuer ce travail pratique. Comme nous l'avons déjà dit dans le TP01, le fait de coder quelque chose de physique est assez satisfaisant. Le dernier exercice nous a donné un peu de fil à retordre mais nous y sommes parvenus.

Nous n'avons pas implémenté de tests unitaires car nous n'avions pas d'implémentation pertinente dans le cadre de nos exercices.

Nous avons passé environ 3**h** chacun à la maison en plus des heures mises à dispositions durant le TP.