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

2021/2022

Informatique Embarquée

Rapport 01

Travail Pratique 1 : Prise en main et thermomètre numérique

Ecole	Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR)
Branche	Informatique embarquée
Etudiants	Barras Simon, Terreaux Nicolas
Groupe	A1
Classe	ISC-IL-2d
Professeur	Bovet Patrick
GitLab	https://gitlab.forge.hefr.ch/embsys/2021-2022/classe-patrick-bovey/gra1/tp01
Version/Date	V1 du 23.02.2022

1 Introduction

Ce premier TP consiste à réaliser un thermomètre qui affiche la température sur un 7-segments. Nous devons utiliser la même cible que dans le cours « Architecture des ordinateurs » mais avec MBed OS 6.

Dans un premier temps, il faut mettre en place l'environnement pour mener à bien ce TP ainsi que tous les suivants. Il faut comprendre le fonctionnement du thermomètre ainsi que du 7-segment en mettant en œuvre plusieurs bus de communication (I^2C et SPI).

2 Résumés

Ce chapitre comprend le résumé des différentes notions apprises durant le TP classé de la manière suivante : non-acquis, à exercer ou parfaitement acquis.

2.1 Non Acquis

Nous ne pensons pas avoir de choses non-acquis.

2.2 À exercer

I^2C, SPI, UART :

Pour ces 3 thèmes, nous avons compris le concept mais il faudrait un peu plus approfondir et pratiquer pour se sentir vraiment à l'aise avec les différentes implémentations.

C++ :

Ce langage est encore assez nouveau pour nous et donc il nous faut plus de pratique pour bien assimiler le maximum de choses.

2.3 Parfaitement acquis

CI/CD :

Nous avons bien compris le fonctionnement et nous sommes à l'aise avec ce qui était demandé pour le CI/CD. Il y a encore énormément de choses à apprendre sur ce thème mais nous sommes à l'aise pour réaliser ce que nous devions faire pour ce TP.

Tests unitaires :

Nous avons mis en place des tests unitaires lancé par le CI/CD pour les lancer sur le rack de test qui se trouve à l'école. Nous pensons que nos tests sont utiles et nous avons bien compris le fonctionnement.

3 Points importants

Ce chapitre comprend tous les points qui nous semblent importants et que nous devons retenir.

- **I^2C**
- **SPI**
- **UART**
- **Thermomètre**
- **CI/CD**
- **C++**

- **Tests unitaires**
- **MBed OS**

4 Questions

Quelle est la vitesse de transmission du bus I²C ?

Comme nous avons vu sur Wikipedia voici un résumé des différentes vitesses en fonction des modes d'I²C :

I²C modes

Mode[12]	Maximum speed	Maximum capacitance	Drive	Direction
Standard mode (Sm)	100 kbit/s	400 pF	Open drain*	Bidirectional
Fast mode (Fm)	400 kbit/s	400 pF	Open drain*	Bidirectional
Fast mode plus (Fm+)	1 Mbit/s	550 pF	Open drain*	Bidirectional
High-speed mode (Hs)	1.7 Mbit/s	400 pF	Open drain*	Bidirectional
High-speed mode (Hs)	3.4 Mbit/s	100 pF	Open drain*	Bidirectional
Ultra-fast mode (UFm)	5 Mbit/s	?	Push–pull	Unidirectional

Figure 1 https://en.wikipedia.org/wiki/I%C2%B2C

Quels sont les avantages et les inconvénients du bus I²C par rapport au bus SPI ?

D'après le site altium.com, voici nos observations :

Avantages d'I²C :

- Nécessite que de quelques pistes de communication
- Prend moins de places

Désavantages d'I²C :

- Plus lent
- Consomme plus d'énergie

Est-ce que le bus I²C est synchrone ou asynchrone ?

Synchrone

Est-ce que le bus I²C fonctionne en "simplex", "full-duplex" ou en "half-duplex" ?

Half-duplex

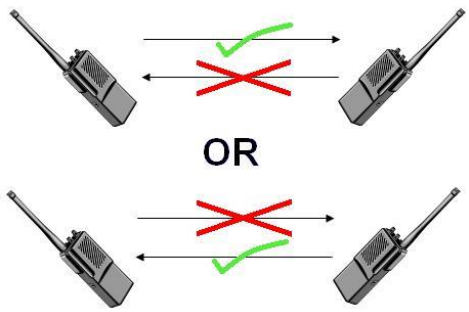

Figure 2 https://fr.wikipedia.org/wiki/Duplex_(canal_de_communication)

Quelle distance peut-on atteindre avec cette interface si on utilise la couche physique RS-485 ?

Maximum 1200 mètres

Quels sont les avantages et les inconvénients l'interface UART par rapport au bus I²C et SPI ?

D'après le site « seedstudio », nous avons fait les conclusions suivantes :

Avantages de UART :

- Il n'y a pas besoin de clock
- Il y a un Bit de parité pour permettre la vérification des erreurs

Désavantages de UART :

- Faibles vitesses de transmission des données
- La taille de la trame de données est limitée à 9 bits seulement.

Est-ce que l'interface UART fonctionne en "simplex", "full-duplex" ou en "half-duplex" ?

Il peut être les 3.

Quelle classe de Mbed OS permet d'utilise l'interface UART ?

UARTSerial.h

5 Conclusion

Nous avons apprécié réaliser ce travail. L'utilisation d'un nouveau module est assez sympathique. Un de nous a cependant perdu énormément de temps pour faire fonctionner l'environnement et nous avons donc dû travailler un peu plus en dehors des heures de cours mais ça n'a pas été trop dérangeant.

Nous sommes satisfaits de notre travail et nous sommes prêts à attaquer les prochains travaux.

Nous avons passé chacun environ **4h** en dehors des cours.

6 Code source

Voici les principales sources de notre travail pratique.

6.1 Main

```
#include "click_thermo.hpp"
#include "mbed.h"
#include "mbed_trace.h"
#include "seven_segments.hpp"

#if defined(MBED_CONF_MBED_TRACE_ENABLE)
#define TRACE_GROUP "Main"
#endif  // MBED_CONF_MBED_TRACE_ENABLE

static constexpr int kLedOn  = 0;
static constexpr int kLedOff = 1;

int main()
```

```
{
#if defined(MBED_CONF_MBED_TRACE_ENABLE)
    mbed_trace_init();
#endif
    // Create a SevenSegments object
    SevenSegments sSegs(1);
    ClickThermo thermo;
    DigitalOut led(LED4, kLedOff);
    sSegs.SwitchOn();

    while (true) {
        double tmp = thermo.GetTemperature();

        sSegs = tmp;
        printf("Temp = %lf\n", tmp);
        ThisThread::sleep_for(100ms);
    }
    return 0;
}
```

6.2 Seven_segments.cpp

```
#include "seven_segments.hpp"

#include <math.h>

#include "mbed.h"
#include "mbed_trace.h"

#if defined(MBED_CONF_MBED_TRACE_ENABLE)
#define TRACE_GROUP "SevenSegments"
#endif  // MBED_CONF_MBED_TRACE_ENABLE

constexpr int kDash       = 0b10000000;
constexpr int kDashDotted = 0b10000001;
const uint8_t kPatterns[] = {
    0b0111111,  // 0
    0b0000101,  // 1
    0b1011011,  // 2
    0b1001111,  // 3
    0b1100101,  // 4
    0b1101110,  // 5
    0b1111110,  // 6
    0b0000111,  // 7
    0b1111111,  // 8
    0b1101111,  // 9
    0b1110111,  // A
    0b1111000,  // b
```

```
    0b0111010,  // C
    0b1011101,  // d
    0b1111010,  // E
    0b1110010,  // F
};

// 0b mid top-left bottom-left bottom bottom-right top top-right

SevenSegments::SevenSegments(int place)
    : display(PA_7, PA_6, PA_5),
      brightness(place == 0 ? PF_10 : PF_3),
      reset(place == 0 ? PC_3 : PC_4),
      latch(place == 0 ? PB_8 : PA_15)
{
    tr_info("Seven Segments at place %d", place);
    switch (place) {
        case 1:
            break;
        case 0:
            tr_warning("Not implemented yet");
            break;
        default:
            tr_error("Place not available");
            break;
    }

    this->display.format(8, 0);
    this->display.frequency(1000000);
    this->brightness = 1;
    this->reset      = 1;
    this->latch      = 0;
}

void SevenSegments::SwitchOn()
{
    this->brightness = 1;
    Send();
    tr_debug("Switch on");
}

void SevenSegments::SwitchOff()
{
    this->brightness = 0;
    Send();
    tr_debug("Switch off");
}

void SevenSegments::Print(int number)
```

```
{
    tr_debug("Print integer %i", number);
    if (number < -9 || number > 99) {
        tr_warning("Number not available");
        PrintError(true);
        return;
    }

    if (number < 0) {
        tr_debug("Negative number");
        PrintDigit(-number);
        this->display.write(kDash);
    } else {
        for (int i = 0; i < 2; i++) {
            int digit = number % 10;
            number /= 10;
            PrintDigit(digit);
        }
    }

    Send();
}

void SevenSegments::Print(double number)
{
    tr_debug("Print double %f", number);
    if (number <= -9.5 || number >= 99.5) {
        tr_warning("Number not available");
        PrintError(false);
        return;
    }

    int unit = number;
    bool dot = (number - unit) != 0;
    if (unit < 0) {
        tr_debug("Negative number");
        PrintDigit(round(-number), (number - unit) != 0);
        this->display.write(kDash);
    } else if (unit < 10) {
        unit      = round(number * 10);
        int digit = unit % 10;
        unit /= 10;
        PrintDigit(digit, false);
        digit = unit % 10;
        PrintDigit(digit, true);
    } else {
        unit      = round(number);
        int digit = unit % 10;
```

```
        unit /= 10;
        PrintDigit(digit, dot);
        digit = unit % 10;
        PrintDigit(digit, false);
    }

    Send();
}

void SevenSegments::PrintError(bool integer = true)
{
    tr_debug("Print error");
    this->display.write(kDash);
    if (integer) {
        this->display.write(kDash);
    } else {
        this->display.write(kDashDotted);
    }
    Send();
}

void SevenSegments::PrintDigit(int digit, bool dot)
{
    tr_debug("Print digit %i, dotted: %i", digit, dot);
    if (digit == 10) {
        this->display.write(0);
    } else {
        int element = kPatterns[digit] << 1 | dot;
        tr_debug("%i", element);
        this->display.write(element);
    }
}

void SevenSegments::Send()
{
    this->latch = 1;
    this->latch = 0;
    tr_debug("Send");
}

void SevenSegments::operator=(int number) { Print(number); }

void SevenSegments::operator=(double number) { Print(number); }
```

6.3 Click_thermo.cpp

```
#include "click_thermo.hpp"
```

```
#include "mbed.h"
#include "mbed_trace.h"

const int addr7bit  = 0x48;       // 7-bit I2C address
const int addr8bit  = 0x48 << 1;  // 8-bit I2C address, 0x90
const int frequency = 4000;
const auto SDA = A4;
const auto SCL = A5;

ClickThermo::ClickThermo() : i2c_(SDA, SCL) {
i2c_.frequency(frequency); }

double ClickThermo::GetTemperature()
{
    char cmd[2];
    cmd[0] = 0x01;
    cmd[1] = 0x00;
    i2c_.write(addr8bit, cmd, 2);
    // read temperature register
    cmd[0] = 0x00;
    i2c_.write(addr8bit, cmd, 1);
    i2c_.read(addr8bit, cmd, 2);
    double tmp = (double((cmd[0] << 8) | cmd[1]) / 256.0);
    return tmp;
}
```

6.4 Test_compile.cpp

```
#include <string.h>
#include <unity.h>

#include <iostream>

#include "click_thermo.hpp"
#include "mbed.h"
#include "mbed_trace.h"
#include "seven_segments.hpp"
void thermo_test(void)
{
    ClickThermo cl;

    double currentTemp1 = cl.GetTemperature();

    printf("Temperature 1 =  %lf\n", currentTemp1);
    ThisThread::sleep_for(500ms);
    TEST_ASSERT_EQUAL(10 < currentTemp1, true);
    TEST_MESSAGE("Temperature is greater than 10 degrees");
    TEST_ASSERT_EQUAL(currentTemp1 < 35, true);
```

```
    TEST_MESSAGE("Temperature is lower than 10 degrees");
    ThisThread::sleep_for(5000ms);

    double currentTemp2 = cl.GetTemperature();
    printf("Temperature 2 =  %lf\n", currentTemp2);
    ThisThread::sleep_for(500ms);

    TEST_ASSERT_EQUAL(10 < currentTemp2, true);
    TEST_MESSAGE("Temperature 2 is greater than 10 degrees");
    TEST_ASSERT_EQUAL(currentTemp2 < 35, true);
    TEST_MESSAGE("Temperature 2 is lower than 10 degrees");
    ThisThread::sleep_for(500ms);

    double diffTemp = currentTemp1 - currentTemp2;
    printf("Difference =  %lf\n", diffTemp);
    ThisThread::sleep_for(500ms);

    TEST_ASSERT_EQUAL(diffTemp * diffTemp < 10, true);
    TEST_MESSAGE(
        "The temperature difference between the first and second
measurement is less than 3 "
        "degrees");
}

void display_test(void)
{
    double kTests[] = {-9.5, -9.4, -4.8, 0, 0.39, 7.21, 12.3, 99.43,
99.5};

    SevenSegments sSegs(1);

    for (int i = 0; i < sizeof(kTests) / sizeof(kTests[0]); i++) {
        sSegs = kTests[i];
        printf("Test: %lf\n", kTests[i]);
        ThisThread::sleep_for(5000ms);
    }

    TEST_ASSERT_EQUAL(true, true);
}
int main()
{
    wait_us(2000000);
    UNITY_BEGIN();
    RUN_TEST(thermo_test);
    RUN_TEST(display_test);
    UNITY_END();
    while (1) {
        asm volatile("nop");
```

```
        }
}
```