Eindopdracht 2

In deze tweede eindopdracht worden de volgende drie leerdoelen van EMS30 getoetst, zie cursushandleiding:

De student is in staat om:

LD4 gebruik te maken van objectgeoriënteerde programmeertechnieken in C++ zoals encapsulation, templates, overerving en polymorfisme zodanig dat de student programmatuur kan ontwikkelen die uitbreidbaar, aanpasbaar en herbruikbaar is;

LD5 op grond van zijn/haar kennis van de big-O-notatie effectief gebruik te maken van de standaard in C++ aanwezige datastructuren en algoritmen;

LD6 een objectgeoriënteerd ontwerp te maken van een programma in UML door middel van usecase-, klassen-, sequentie-, toestand- en activiteitendiagrammen.

In totaal kun je 100 punten behalen voor deze eindopdracht verdeeld over de leerdoelen volgens de weging die vermeld is in de cursushandleiding.

Leerdoel LD4 wordt getoetst met opdracht E2.1 tot en met opdracht E2.4 en hiermee zijn 60 punten te behalen. Leerdoel LD5 wordt getoetst met opdracht E2.5 en hiermee zijn 20 punten te behalen. Leerdoel LD6 wordt getoetst met opdracht E2.6 en hiermee zijn 20 punten te behalen.

De deadline voor deze eindopdracht is, zoals in de cursushandleiding vermeld: **zondag 25 juni 2023, 23.59 uur**.

Het op te leveren werk bestaat uit nieuwe code op de EMS30 repository en uit een bondig verslag. Commit en push een .pdf-versie van het verslag in je repository. Bij de opgaven zal worden aangegeven als er verslaglegging wordt verwacht. *N.B. zorg ervoor dat er tenminste 1 commit per gemaakte programmeeropdracht is.*

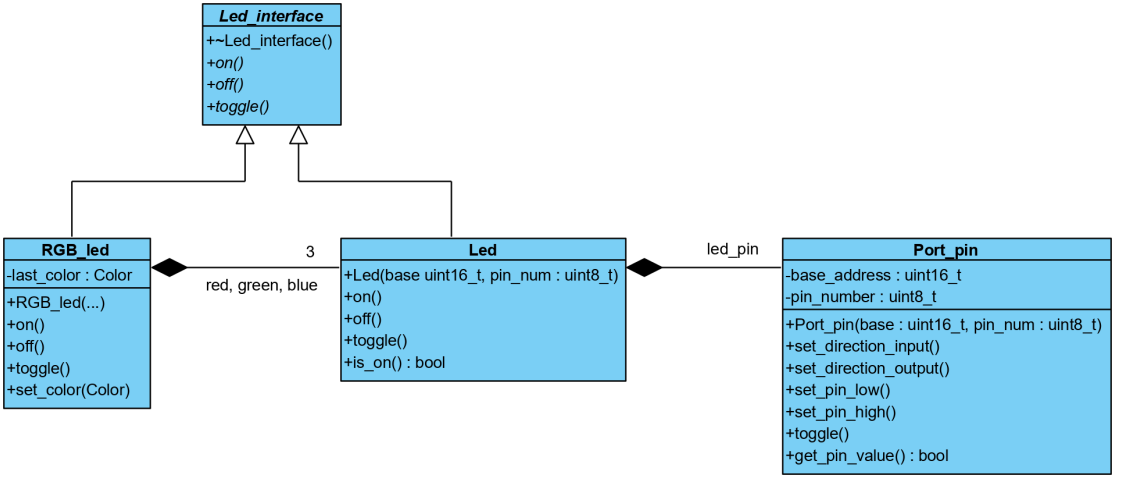
Leerdoel 4 (60 punten)

In opdracht E2.1 tot en met opdracht E2.4 gaan we een aantal simpele *classes* bouwen in C++ op het MSP430G2ET Launchpad platform[1]. We gebruiken hierbij de MSP430G2553

[1] We gaan ervan uit dat je een Launchpad MSP-EXP430G2ET hebt. Als je nog een oude MSP-EXP430G2 hebt, dan zul je zelf een RGB-led op een breadboard moeten plaatsen. Als je niet in bezit bent van een RGB-led, dan kun je die ophalen in de docentenkamer.

microcontroller en leds op de Launchpad zelf. Je hebt dus geen breadboard nodig voor deze opdracht. Let er wel op dat je de MSP430G2553 en de jumpers op de Launchpad (terug)zet.

De classes gaan we zo opbouwen dat ze gemakkelijk te gebruiken zullen zijn op andere platformen en dat de interface zo leesbaar mogelijk is. De opbouw van de classes is te zien in figuur 1.



Figuur 1: UML klassendiagram van opdracht E2.1 tot en met opdracht E2.4

We beginnen met de `Port_pin` class. Het doel is om deze class zodanig op te zetten dat deze class op verschillende microcontrollers van verschillende fabrikanten te gebruiken is. Om dit te bereiken willen we niet de standaard register definities van de fabrikanten gebruiken, maar zelf op basis van adressen de registers benaderen. De GPIO-schakelingen werken namelijk vrijwel altijd hetzelfde: Je hebt een directie register, om een pin als input of output in te stellen en je hebt een output en input register om de pinwaarde respectievelijk te schrijven en te lezen.

De `Port_pin` class heeft een basisadres *Engels: base address*. Dit is het eerste (laagste) geheugenadres van de port peripheral. Bij de MSP430G2553 is bijvoorbeeld het basisadres van PORT1 gelijk aan 0x20 wat correspondeert met het P1IN register. Bij PORT2 is dit 0x28 wat correspondeert met het P2IN register. Het meegeven van een adres is handig omdat van daaruit kan worden bepaald waar het input, output en directie register is te vinden. Het output register heeft bijvoorbeeld een offset van 1 ten opzichte van het basisadres. Dus P1OUT staat op 0x20 + 1 en P2OUT staat op 0x28 + 1.

In de taal C zou je een adres kunnen benaderen met behulp van pointer casting:

```
*((volatile uint8_t *) (0x21) ) ^= (1<<2); //P1.2 toggle
```

In C++ is het netter om dit te doen met de **reinterpret_cast** template:

```
*reinterpret_cast<volatile uint8_t*>(0x21) ^= (1<<2); //P1.2 toggle
```

E2.1 Maak een nieuw project aan in Code Composer Studio, binnen jouw EMS30 repository. Hernoem `main.c` naar `main.cpp` zodat CCS automatisch de C++-compiler zal gebruiken. Vul dit met de code van listing 1.

```
#include <cstdint>
#include <msp430.h>
#include "msp430_reg.h"

int main() {
    //watchdog timer uitzetten
    WDTCTL = WDTPW | WDTHOLD;
    //de I/O-class Port_pin gebruiken!
    Port_pin led1 {PORT1, 0}; //led op P1.0
    led1.set_direction_output();
    Port_pin s2 {PORT1, 3}; //knop op P1.3 met externe pull up
    s2.set_direction_input();
    while(1) {
        led1.toggle();
        while(s2.get_pin_value() == false) {
            //niets doen
        }
        __delay_cycles(60000);
    }
}
```

Listing 1: Main code voor opdracht E2.1

Download zelf het msp430_reg.h bestand en voeg dit toe aan je project. Dit bestand bevat de definities van de basis-adressen en offsets die je kunt gebruiken bij het implementeren en gebruiken van de classes.

Implementeer vervolgens de class `Port_pin`. Gebruik de hierboven besproken **reinterpret_cast** om het register achter een adres te benaderen. De benodigde basisadressen en offsets kun je vinden in de headerfile msp430_reg.h.

Tip: Zie dictaat C++ paragraaf 2.5 voor meer informatie over het initialiseren van datavelden.

Breid het programma uit listing 1 uit zodat de class `Port_pin` volledig wordt getest. Neem de declaratie van de class `Port_pin` op in een apart bestand `port_pin.h` en neem de definities van de memberfuncties op in een apart bestand `port_pin.cpp` zodat deze class eenvoudig in andere projecten gebruikt kan worden.

E2.2 Implementeer nu de classes `Led_interface` en `Led`. Vervang de code in `main` door de code die gegeven is in listing 2. Zorg ervoor dat binnen de constructor de pin als output wordt ingesteld en laag wordt gemaakt. Maak hiervoor gebruik van een `Port_pin` object.

```
#include <cstdint>
#include <msp430.h>
#include "msp430_reg.h"

int main() {
    //watchdog timer uitzetten
    WDTCTL = WDTPW | WDTHOLD;
    //de I/O-class Led gebruiken!
    Led led1 {PORT1, 0}; //led op P1.0
    //de I/O-class Port_pin gebruiken!
    Port_pin s2 {PORT1, 3}; //knop op P1.3 met externe pull up
    s2.set_direction_input();
    while(1) {
        led1.toggle();
        while(s2.get_pin_value() == false) {
            //niets doen
        }
        __delay_cycles(60000);
    }
}
```

Listing 2: Main code voor opdracht E2.2

Breid het programma uit listing 2 uit zodat de class `Led` volledig wordt getest. Neem de declaratie van de class `Led_interface` op in een apart bestand `led_interface.h` Neem de declaratie van de class `Led` op in een apart bestand `led.h` en neem de definities

van de memberfuncties op in een apart bestand `led.cpp` zodat deze class eenvoudig in andere projecten gebruikt kan worden.

E2.3 Implementeer nu ook de class `RGB_led`. Met behulp van de memberfunctie `void RGB_led::set_color(Color c)` kun je de kleur van de RGB-led instellen. Het parametertype `Color` is een `enum` die *in* de class `RGB_led` is gedefinieerd:

```
enum Color {Blue = 1, Green, Cyan, Red, Magenta, Yellow, White};
```

Als de memberfunctiefunctie `void RGB_led::toggle()` aangeroepen wordt dan moet de RGB-led uitgezet worden als de led brandt en anders moet de led aangezet worden in de kleur die is ingesteld met de memberfunctie `void RGB_led::set_color(Color c)`. Vervang de code in `main` door de code die gegeven is in listing 3.

```
#include <cstdint>
#include <array>
#include <msp430.h>
#include "msp430_reg.h"

// polymorph function
void blink_n_times(Led_interface& led, unsigned int n) {
    for (int i{0}; i < n*2; ++i) {
        led.toggle();
        __delay_cycles(600000);
    }
}

int main()
{
    //watchdog timer uitzetten
    WDTCTL = WDTPW | WDTHOLD;
    Led led1 {PORT1, 0}; //led op P1.0
    blink_n_times(led1, 5);
    Led led2 {PORT1, 6};
    blink_n_times(led2, 5);

    RGB_led led3 {PORT2, 1, PORT2, 3, PORT2, 5};
    std::array<RGB_led::Color, 7> all_colors {
        RGB_led::Blue, RGB_led::Green, RGB_led::Cyan, ↩
 ↪ RGB_led::Red,
        RGB_led::Magenta, RGB_led::Yellow, RGB_led::White
```

```
    };
    for (auto c: all_colors) {
        led3.set_color(c);
        __delay_cycles(600000);
    }

    led3.set_color(RGB_led::White);
    __delay_cycles(600000);
    blink_n_times(led3, 5);
    led3.set_color(RGB_led::Red);
    blink_n_times(led3, 1);
    led3.set_color(RGB_led::Green);
    blink_n_times(led3, 2);
    led3.set_color(RGB_led::Blue);
    blink_n_times(led3, 3);

    led3.set_color(RGB_led::Magenta);
    while (1) {
        led3.toggle();
        __delay_cycles(60000);
    }
}
```

Listing 3: Main code voor opdracht E2.3

Dit programma moet achtereenvolgens:

- de groene led op pin P1.0 vijf keer laten knipperen;
- de rode led op pin P1.6 vijf keer laten knipperen;
- de RGB-led op pinnen P2.1, P2.3 en P2.5 alle kleuren laten doorlopen (blauw, groen, cyaan, rood, magenta, geel, wit);
- de RGB-led vijf keer laten knipperen in de kleur wit;
- de RGB-led één keer laten knipperen in de kleur rood;
- de RGB-led twee keer laten knipperen in de kleur groen;
- de RGB-led drie keer laten knipperen in de kleur blauw;
- de RGB-led continue snel laten knipperen in de kleur magenta;

Neem de declaratie van de class `RGB_led` op in een apart bestand `RGB_led.h` en neem de definities van de memberfuncties op in een apart bestand `RGB_led.cpp` zodat deze class eenvoudig in andere projecten gebruikt kan worden.

Onze eenvoudige I/O-classes zijn redelijk portable tussen verschillende microcontrollers. Het enige wat aangepast moet worden is de headerfile met de basisadressen en de offsets van de registers. Maar wat als we dit zouden gebruiken op de CC3220S? De geheugenadressen zijn niet langer 16 bits, en de registers zijn ook niet langer 8 bits!

E2.4 Gebruik templates om mee te kunnen geven van welke type de geheugenadressen zijn en van welk type de registers zijn.

Leerdoel 5 (20 punten)

E2.5 Voer de opdrachten van week 7 les 2 uit (als je dat nog niet hebt gedaan). Zorg ervoor dat de code die je hebt geschreven goed vindbaar is in jullie repository en rapporteer al jullie bevindingen in het verslag.

Leerdoel 6 (20 punten)

E2.6 Jullie hebben hoogstwaarschijnlijk al eerder een softwareontwerp gemaakt tijdens het PEE50-project of tijdens de stage. Kies een van de softwareontwerpen die één van jullie twee al eerder tijdens de stage of het PEE50-project heeft gemaakt en herontwerp dit op een objectgeoriënteerde manier m.b.v. UML. Je hoeft alleen het ontwerp maar te modelleren door middel van UML-diagrammen. Het is dus niet nodig om C++-code te schrijven. Neem contact op met Roy Bakker BaRoy@hr.nl, als je geen geschikt project hebt om te modelleren.

Neem in je verslag een korte beschrijving op van het betreffende project en de requirements voor de software. Neem daarna de volgende UML-diagrammen op in je verslag:

- usecasediagram;
- klassendiagram;
- minstens één sequentiediagram;
- minstens één toestandsdiagram;
- minstens één activiteitendiagram.

Let er goed op dat de diagrammen onderling consistent zijn.