Eindopdracht H2

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 EH2.1 tot en met opdracht EH2.3 en hiermee zijn 60 punten te behalen. Leerdoel LD5 wordt getoetst met opdracht EH2.4 tot en met opdracht EH2.8 en hiermee zijn 20 punten te behalen. Leerdoel LD6 wordt getoetst met opdracht EH2.9 en hiermee zijn 20 punten te behalen.

De deadline voor deze eindopdracht is, zoals in de cursushandleiding vermeld: **zondag 9 juli, 23.59 uur**.

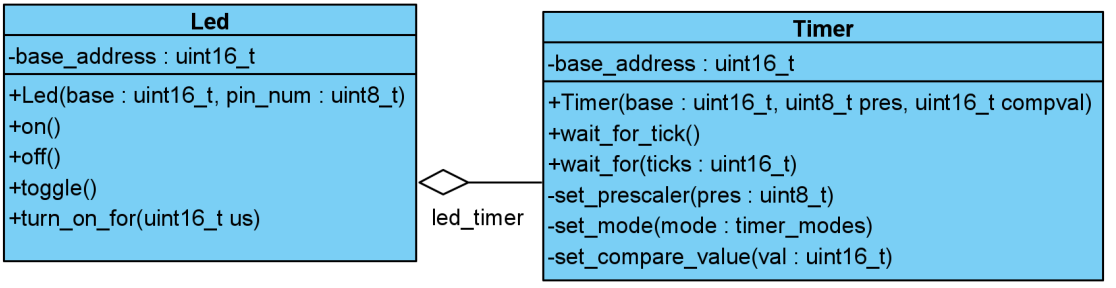
Het op te leveren werk bestaat uit nieuwe code op de EMS30 repository en uit een bondig verslag. Commit en push een .pdf-versie van het verslag in je repository. Bij de opgaven zal worden aangegeven als er verslaglegging wordt verwacht.

Leerdoel 4 (60 punten)

In opdracht EH2.1 tot en met opdracht EH2.3 gaan we een simpele *Timer*- en *Led-class* bouwen in C++ op het MSP430G2 Launchpad platform.

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

We bekijken de `Timer` class. Het doel is om deze class zodanig op te zetten zodat het op verschillende microcontrollers van verschillende fabrikanten is toe te passen. Om dit te



Figuur 1: UML class diagram van opdracht EH2.1 tot en met opdracht EH2.3

bereiken willen we niet de standaard register definities van de fabrikanten gebruiken, maar zelf op basis van adressen de registers benaderen. Timer schakelingen werken namelijk vrijwel altijd hetzelfde: Je hebt een *control* register om dingen als de prescaler en klokbron in te stellen en er is tenminste één *compare* register.

De `Timer` class heeft een *base adres*. Dit is het eerste (laagste) geheugenadres van de peripheral. Bij de MSP430G2553 heeft bijvoorbeeld TIMER0 het laagste adres van 0x160 wat correspondeert met het TA0CTL register. Bij TIMER1 is dit 0x180 wat correspondeert met het TA1CTL register. Het meegeven van een adres is handig omdat van daaruit kan worden bepaald waar het control en compare register is te vinden. Het compare register heeft bijvoorbeeld een offset van 0x12 ten opzichte van het basisadres. Dus TA0CCR0 staat op 0x160 + 0x12 en TA1CCR0 staat op 0x180 + 0x12.

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

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

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

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

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_herk.h"

int main() {
```

```
    //watchdog timer uitzetten
    WDTCTL = WDTPW | WDTHOLD;
    //DCO op 16MHz
    DCOCTL = 0;
    BCSCTL1 = CALBC1_16MHZ;
    DCOCTL  = CALDCO_16MHZ;
    //led op P1.0
    P1DIR |= 1;

    while(1) {
        P1OUT ^= 1; //P1.0 toggle
        //~1000 ms wachten
        __delay_cycles(160000*1000);
    }
}
```

Listing 1: Main code om mee te beginnen

Download zelf het msp430_reg_herk.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 van de Timer-class.

EH2.1 Implementeer nu de class `Timer`. Deze timer zal geen gebruik maken van interrupt functies. Gebruik de hierboven besproken **`reinterpret_cast`** om het register achter een adres te benaderen. De timer zal voor het gemak alleen gebruik maken van compare register 0.

Om de hele timer perfect portable te krijgen met andere microcontrollers is in dit geval te lastig en valt buiten de scope van deze opdracht. Wel wordt verwacht dat de class bruikbaar is met elke timer binnen de msp430G2553. Zorg ervoor dat binnen de constructor de timer op UP-mode wordt gezet. Het `timer_mode` type is een enumeratie die gedefinieerd is in de headerfile: msp430_reg_herk.h. De `wait_for()` functie kan simpelweg tellen hoe vaak de compare interrupt flag (*tick*) is geset. Deze functie werkt dus door te *pollen*. Vergeet niet om de compare interrupt flag te resetten.

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

Listing 2 laat door softwarematige PWM de led dimmen. Gebruik deze code om jouw Timer implementatie te testen.

```
#include <cstdint>
#include <msp430.h>
#include "msp430_reg_herk.h"
#include "timer.h"

int main() {
    //watchdog timer uitzetten
    WDTCTL = WDTPW | WDTHOLD;
    //DCO op 16MHz
    DCOCTL = 0;
    BCSCTL1 = CALBC1_16MHZ;
    DCOCTL  = CALDCO_16MHZ;
    //led op P1.0
    P1DIR |= 1;
    //prescaler van 8 => 2Mhz
    //compare tick elke 1000 us
    Timer timer0 {TIMER0, 8, 2000};
    //elke us een tick.
    Timer timer1 {TIMER1, 1, 16};

    //Onderstaande code laat de led dimmen door de
    //duty cycle aan te passen. f=1kHz. Max duty = 60%
    std::uint16_t duration_in_us {0};
    bool going_up {true};
    while (1) {
        //wachten op einde ms tick.
        timer0.wait_for_tick();
        //led aan voor aantal us
        P1OUT |= 1;
        timer1.wait_for(duration_in_us);
        P1OUT &= ~1;
        //hysteresis tussen 0 us en 600 us
        if (duration_in_us > 599)
            going_up = false;
        if (duration_in_us < 1)
            going_up = true;
        //change duty
        going_up ? duration_in_us++ : duration_in_us--;
```

```
    }
}
```

Listing 2: Main code voor opdracht EH2.1

EH2.2 Implementeer nu de class Led. Ook deze class maakt gebruik van een basis adres met offsets, om met **reinterpret_cast** het juiste register te benaderen.

Binnen de class Led kun je gebruik maken van een Timer object met een ticktijd van 1 µs. De turn_on_for() functie moet hiervan gebruikmaken.

Vervang de code in main.c met listing 3 om jouw class te testen.

```
#include <cstdint>
#include <msp430.h>
#include "msp430_reg_herk.h"
#include "timer.h"
#include "led.h"

int main() {
    //watchdog timer uitzetten
    WDTCTL = WDTPW | WDTHOLD;
    //DCO op 16MHz
    DCOCTL = 0;
    BCSCTL1 = CALBC1_16MHZ;
    DCOCTL  = CALDCO_16MHZ;
    //led op P1.0
    Led led1 {PORT1, 0};
    //prescaler van 8 => 2Mhz
    //compare tick elke 1000 us
    Timer timer0 {TIMER0, 8, 2000};

    //Onderstaande code laat de led dimmen door de
    //duty cycle aan te passen. f=1kHz. Max duty = 60%
    std::uint16_t duration_in_us =0;
    bool going_up {true};

    while (1) {
        //wachten op einde ms tick.
        timer0.wait_for_tick();
        //led aan voor aantal us
```

```
            led1.turn_on_for(duration_in_us);
            //hysteresis tussen 0us en 600us
            if(duration_in_us > 599)
                going_up = false;
            if(duration_in_us < 1)
                going_up = true;
            //change duty
            going_up ? duration_in_us++ : duration_in_us--;
        }
    }
```

Listing 3: Main code voor opdracht EH2.2

Onze eenvoudige 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, de timerregisters zijn ook niet langer 16 bits en de GPIO-registers zijn niet langer 8 bits!

EH2.3 Gebruik templates om mee te kunnen geven van welke type de geheugenadressen zijn en van welk type de registers zijn.

Leerdoel 5 (20 punten)

In het voorbeeldprogramma `timedsort2.cpp` is de class `Stopwatch` opgenomen. Deze class kan gebruikt worden om tijd te meten. In het voorbeeldprogramma wordt de tijd gemeten die het `selection_sort` algoritme nodig heeft om een vector met n elementen te sorteren.

EH2.4 Bepaal de complexiteit van het `selection_sort` algoritme in big-O-notatie door de source code te analyseren. Stel vervolgens de orde van dit algoritme empirisch vast door de benodigde tijd te meten voor minstens 10 verschillende waarden van n. Dit kun je natuurlijk automatiseren door het programma aan te passen. Maak een grafiek met behulp van Excel. In Excel kun je een zogenaamde trendlijn bepalen die de gemeten waarden zo goed mogelijk (met een bepaalde functie) benaderd. Geef de gevonden functie en de waarde van R^2 (die aangeeft hoe goed de trendlijn de meetwaarden benaderd) ook weer in de grafiek. Wat is de gemeten complexiteit van dit algoritme

uitgedrukt in big-O-notatie? Komt dit overeen met de complexiteit die is gevonden door de source code te analyseren?

EH2.5 In het programma is ook een implementatie van het `heap_sort` algoritme gegeven. Vervang de aanroep naar `selection_sort` door een aanroep naar `heap_sort`. Bepaal de werking en de complexiteit van het `heap_sort` algoritme in big-O-notatie door literatuuronderzoek te doen en door de source code te analyseren. Leg de werking van de gegeven implementatie in jullie eigen woorden uit in het verslag. Vermeld de gebruikte bronnen in jullie verslag. Stel vervolgens de orde van dit algoritme empirisch vast door de benodigde tijd te meten voor minstens 10 verschillende waarden van n. Maak een grafiek met behulp van Excel. Bepaal ook welke trendlijn het beste past (de hoogste waarde van R^2 heeft). Wat is de gemeten complexiteit van dit algoritme uitgedrukt in big-O-notatie? Komt dit overeen met de complexiteit die is gevonden door de source code te analyseren?

EH2.6 Omdat de elementen die we sorteren bestaan uit integers in een beperkte range (minimaal 0 en maximaal 999) kan een sneller sorteeralgoritme gebruikt worden. Dit algoritme wordt `counting_sort` genoemd en werkt als volgt:

- Maak een `array` genaamd `count` aan met 1000 integers en vul die met nullen.
- Voor elk elke element `e` uit de te sorteren `vector` doe: `++count[e]`.
- Vul nu de vector met count[0] nullen, count[1] enen, count[2] tweeën, enz. De vector is nu gesorteerd!

Implementeer dit algoritme in het gegeven programma. Toon aan dat de implementatie correct werkt. Bepaal de complexiteit van het `counting_sort` algoritme in big-O-notatie door jullie source code te analyseren. Stel vervolgens de orde van dit algoritme empirisch vast door de benodigde tijd te meten voor minstens 10 verschillende waarden van n. Maak een grafiek met behulp van Excel. Bepaal ook welke trendlijn het beste past (de hoogste waarde van R^2 heeft). Wat is de gemeten complexiteit van dit algoritme uitgedrukt in big-O-notatie? Komt dit overeen met de complexiteit die is gevonden door jullie source code te analyseren?

In het voorbeeldprogramma `timedcontains.cpp` is naast de class `Stopwatch` ook een functie genaamd `contains_element_method_1` gegeven. Deze functie gebruikt een zogenoemde lineaire zoekmethode om te bepalen of de integer `e` in de `vector` `v` voorkomt.

EH2.7 Bepaal de complexiteit van de functie `contains_element_method_1` in big-O-notatie door de source code te analyseren. Stel vervolgens de orde van deze functie empirisch vast door de benodigde tijd te meten voor minstens 10 verschillende waarden van n. Bedenk dat de big-O-notatie de *worst case* complexiteit aangeeft. Hoe kun je er voor zorgen dat de functie `contains_element_method_1` de *worst case* executietijd nodig heeft? Maak een grafiek met behulp van Excel. Bepaal ook welke trendlijn het beste past (de hoogste waarde van R^2 heeft). Wat is de gemeten complexiteit van dit algoritme uitgedrukt in big-O-notatie? Komt dit overeen met de complexiteit die is gevonden door de source code te analyseren?

EH2.8 Als we weten dat de `vector` waarin we zoeken gesorteerd is, dan kunnen we gebruik maken van een sneller zoekalgoritme dat binair zoeken wordt genoemd. Dit algoritme werkt als volgt:

- Als de `vector` géén elementen bevat, geef dan **`false`** terug
- Kijk of het middelste element van de `vector` het element is wat je zoekt. Geef **`true`** terug als dit zo is.
- Kijk of het middelste element van de `vector` groter is dan het element wat je zoekt. Herhaal het algoritme op de eerste helft van de `vector`.
- Het middelste element van de `vector` is blijkbaar kleiner dan het element wat je zoekt. Herhaal het algoritme op de tweede helft van de `vector`.

Implementeer dit algoritme in de functie `contains_element_method_2` die gegeven is in het programma. Toon aan dat de implementatie correct werkt door in de eerste regel van `main` de `1` te vervangen door een `2`. Bepaal de complexiteit van de functie `contains_element_method_2` in big-O-notatie door jullie source code te analyseren. Bedenk dat de big-O-notatie de *worst case* complexiteit aangeeft. Hoe kun je er voor zorgen dat de functie `contains_element_method_2` de *worst case* executietijd nodig heeft? Stel vervolgens de orde van deze functie empirisch vast door de benodigde tijd te meten voor minstens 10 verschillende waarden van n. Maak een grafiek met behulp van Excel. Bepaal ook welke trendlijn het beste past (de hoogste waarde van R^2 heeft). Wat is de gemeten complexiteit van dit algoritme uitgedrukt in big-O-notatie? Komt dit overeen met de complexiteit die is gevonden door jullie source code te analyseren?

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)

EH2.9 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.

Let op! Het is *niet* toegestaan om bij deze herkansing dezelfde (of verbeterde) diagrammen als bij de reguliere toets in te leveren. Neem contact op met Roy Bakker BaRoy@hr.nl, als je geen geschikt project (meer) 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.