

RTS10

MINOR EMBEDDED SYSTEEM

HOGESCHOOL ROTTERDAM

Robbin Koot, Max Overvoorde | RTS10 | 11-9-2022

Inhoud

[Inleiding 2](#_Toc113833095)

[Assembly 3](#_Toc113833096)

[Assignment 1 3](#_Toc113833097)

[Assignment 2 3](#_Toc113833098)

[Assembly 3](#_Toc113833099)

[In C 3](#_Toc113833100)

[Assignment 5 4](#_Toc113833101)

[Assembly 4](#_Toc113833102)

[In C 5](#_Toc113833103)

[Assignment 7 5](#_Toc113833104)

[Assembly 5](#_Toc113833105)

[In C 7](#_Toc113833106)

[Resultaten 8](#_Toc113833107)

[Week 2 9](#_Toc113833108)

[Opdracht 1 9](#_Toc113833109)

[Opdracht 2 10](#_Toc113833110)

[Opdracht 3 11](#_Toc113833111)

[Opdracht 4 12](#_Toc113833112)

[Bibliografie 13](#_Toc113833113)

[Bijlage 14](#_Toc113833114)

[Opdracht 1 14](#_Toc113833115)

[Opdracht 2 15](#_Toc113833116)

[Opdracht 3 16](#_Toc113833117)

[Opdracht 4 17](#_Toc113833118)

# Inleiding

Deze opdrachten zijn gemaakt na aanleiding van de minor Embedded systems op de Hogeschool Rotterdam voor het vak RTS10 (Real time systems). In dit document wordt een omschrijving gegeven van de opdrachten en antwoord gegeven op de desbetreffende vragen. Hiernaast zal de gebruikte code als snippets aanwezig zijn. In de bijlage zal ook per opdracht de volledige code te zien zijn. In aanvulling hierop is de assembly code van RTS10 ook te vinden in dit document.

# Assembly

## Assignment 1

Deze code was al gegeven, dus niet relevant voor de opdracht

## Assignment 2

### Assembly



### In C



## Assignment 5

### Assembly

.cpu cortex-m4

.thumb

.syntax unified

**.globl** multiply

**.text**

.thumb\_func

**multiply:**

MOVS.N R2, R1 // 3e variabele wordt b

MOVS.N R1, R0 // 2e variabele wordt a

MOVS.N R0, #0 // m wordt 0

**multiply2:**

CMP.N R2, #0 //--> compare B met 0

BNE.N skip1 //--> Niet gelijk aan 0? skip naar skip1

MOVS.N R0, #0 //--> Wel gelijk aan 0? Move de waarde 0 naar R0 (return)

BX.N LR //--> return naar eerdere laag code

**skip1:**

CMP.N R2, #1 //--> compare B met 1

BNE.N skip2 //--> niet gelijk aan 1? skip naar skip2

ADDS.N R0, R0, R1 //--> wel gelijk aan 1? Tel a bij m op in return waarde

BX.N LR //--> return naar eerdere laag code

**skip2:**

MOVS.N R3, #1 //--> Maak R3 gelijk aan b

ANDS.N R3, R2 //--> bitmask zodat de eerste bit alleen gezien wordt, return op R3

CMP.N R3, #0 //--> is R3 gelijk aan 0?

BNE.N skip3 //--> Nee? skip3, Ja, ga door

LSLS.N R1, R1, #1 //--> Left shift A met 1 bit

LSRS.N R2, R2, #1 //--> right shift B met 1 bit

B.N multiply2 //--> recurse de functie door te springen naar de functienaam opnieuw

**skip3:**

ADDS.N R0, R0, R1 //--> Tel m bij a op

LSLS.N R1, R1, #1 //--> Left shift a met 1

LSRS.N R2, R2, #1 //--> right shift b met 1

B.N multiply2

### In C

/\* main.c simple program to test assembler program \*/

**#include** <stdio.h>

**extern** **int** **multiply**(**int** a, **int** b);

**int** **main**(**void**)

{

**extern** **void** **initialise\_monitor\_handles**(**void**);

initialise\_monitor\_handles();

**int** a = multiply(10505, 35489);

**printf**("Result of test(3, 5) = %d\n", a);

**return** 0;

}

## Assignment 7

### Assembly



### In C



### Resultaten



# Week 2

## Opdracht 1

In deze opdracht sturen we een ledje aan zonder gebruik te maken van een API. Dit gebeurt volledig in C.

In het begin van het programma zijn er definities gemaakt om de adressen van verschillende registers op te slaan. Zo wordt het aansturen van de leds duidelijker en overzichtelijker. Dit is hieronder te zien.



Wat opvalt is het gebruik van de keyword “volatile”. Dit keyword geeft aan dat tijdens het runnen van de code dit adres de mogelijkheid heeft om te veranderen zonder dat het expliciet vermeld hoeft te worden. Dit zorgt ervoor dat de code optimizer niet aan deze waarde kan zitten (Volatile (computer programming), 2021).

wat er bij de listing in de opdracht miste was de laatste #define, die hierboven te zien is. Dit staat voor de output dataregister die bij de GPIOD hoort. Hierbij zie je dat de offset voor de data register gelijk staat aan de waarde 0x14.

## Opdracht 2

In deze opdracht sturen we de leds aan in C met gebruik van de CMSIS API van Arm.

Bijna de volledige code is al gegeven, en hier hoeven alleen nog maar regel 11 en 18 aan toegevoerd te worden. Op regel 11 veranderen we de data van output register 0x5000. Dit zorgt er voor dat bit 12 en bit 14 aangaan, welke gelijkstaan aan de rode en groene LED.

Op regel 18 gebruiken we de XOR operatie om de bits 12 t/m 15 te inverteren.



Als de code van opdracht 2 en 1 vergeleken wordt zie je dat opdracht 2 gebruik maakt van pointers. Hieronder zie je de vertaalde assembly code.

2.1



2.2



Hiermee kunnen we constateren dat beide codes hetzelfde zijn. Het maakt niet uit of je het op de manier van 2.1 of 2.2 doen, want functioneel doen ze hetzelfde.

## Opdracht 3

Tijdens opdracht 3 maken we gebruik van de ingebouwde API van STM32. Deze keer maken we gebruik van de API LL (Low-Layer drivers). Hierbij wordt een UI gebruikt om erg makkelijk de pinnen van de LEDS in te stellen, en de code wordt automatisch gegenereerd. De toegevoegde regel is hieronder te zien



De GPIO\_TogglePin functie toggelt de pin gespecificeerd in de functie. Hierbij van GPIOD de groene en de rode pin. Deze namen worden ook automatisch gegenereerd waardoor het coderen nog een stuk makkelijker gaat.

Tijdens het builden van het programma is in de build analyzer te zien hoeveel ruimte de code nodig heeft. Met de optimizer kan dit veranderd worden. In een niet geoptimaliseerde staat was de .text grootte van opdracht 2.3 2,48KB groot:



Na het optimaliseren van de code gekeken naar de meest efficiënte grootte, is dit 1024B:



## Opdracht 4

In opdracht 4 maken we nu gebruik van de HAL API in plaats van de LL. HAL staat voor “Hardware Abstraction layer” en met HAL kan je op een erg makkelijke manier de verschillende hardware delen aansturen in de chip.

De vermiste code is hieronder terug te vinden:



Hier zie je dat de leds allemaal getoggled worden door deze keer een HAL functie.

In een niet geoptimaliseerde staat was de .text grootte van opdracht 2.4 5,17KB



Na het optimaliseren van de code gekeken naar de meest efficiënte grootte, is dit 3,35KB 

Hieruit is te constateren dat het gebruik maken van een hogere abstractielaag zorgt voor het makkelijker aansturen van bepaalde onderdelen in de chip. Dit heeft wel een nadeel, want dit zorgt ervoor dat het programma veel meer ruimte inneemt.

Hierdoor is het goed om na te denken over welke abstractielaag je wilt gebruiken en hoeveel opslag je daarvoor nodig hebt. Dit zijn eigenschappen en kernmerken die kunnen helpen tijdens het ontwerpen van een systeem of uitkiezen van een specifieke processor.

# Bibliografie

*Volatile (computer programming)*. (2021, October 1). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Volatile\_(computer\_programming)

# Bijlage

## Opdracht 1



## Opdracht 2



## Opdracht 3



## Opdracht 4

