



RTS₁₀

MINOR EMBEDDED SYSTEEM
HOGESCHOOL ROTTERDAM

Inhoud

Inleiding	2
Assembly	3
Assignment 1.....	3
Assignment 2	3
Assembly.....	3
In C.....	3
Assignment 5.....	4
Assembly.....	4
In C.....	5
Assignment 7	5
Assembly.....	5
In C.....	7
Resultaten	8
Week 2.....	9
Opdracht 1	9
Opdracht 2.....	10
Opdracht 3.....	11
Opdracht 4	12
Bibliografie	13
Bijlage	14
Opdracht 1	14
Opdracht 2.....	15
Opdracht 3.....	16
Opdracht 4	17

Inleiding

Deze opdrachten zijn gemaakt na aanleiding van de minor Embedded systems op de Hogeschool Rotterdam voor het vak RTSio (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 RTSio ook te vinden in dit document.

Assembly

ASSIGNMENT 1

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

ASSIGNMENT 2

Assembly

```
.cpu cortex-m4
.thumb
.syntax unified
.globl multiply
.text
.thumb_func
multiply:
    MOVS.N R2, #0                // making variable for return value, R2
= m, R3 = i
check:
    CMP.N R0, #0                // if A = 0
    BNE.N loop                  // als ie niet gelijk is aan 0, jump naar loop
    MOVS.N R0, R2                // return M als return waarde
    BX.N LR                     // weten we niet helemaal zeker
loop:
    ADD.N R2, R2, R1             // m = m + b
    SUBS.N R0, #1               // A = A - 1
    B.N check
```

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(0, 10);
    printf("Result of test(3, 5) = %d\n", a);
    return 0;
}
```

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
                           //--> 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
                           //--> 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?
                           //--> Nee? skip3, Ja, ga door
    BNE.N skip3            //--> Left shift A met 1 bit
    LSLS.N R1, R1, #1       //--> right shift B met 1 bit
    LRS.N R2, R2, #1       //--> recurse de functie door
    B.N multiply2
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
    LRS.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

```

        .cpu cortex-m4
        .thumb
        .syntax unified
        .globl power
        .text
        .thumb_func

multiply:
        MOVS.N R2, #0                // making variable for return value,
R2 = m, R3 = i
check:
        CMP.N R0, #0                // if A = 0
        BNE.N loop                  // als ie niet gelijk is aan 0, jump naar
loop
        MOVS.N R0, R2                // return M als return waarde
        BX.N LR                      // weten we niet helemaal zeker

loop:
        ADD.N R2, R2, R1              // m = m + b
        SUBS.N R0, #1                // A = A - 1
        B.N check

power:
        PUSH.N {LR}
        MOVS.N R2, #1
        CMP.N R1, #0
        BNE.N check_loop
        MOVS.N R0, #1
        BX.N LR                      // volgorde: R0 = n, R1 = m,
R2 = p

check_loop:
        CMP.N R1, #1
        BEQ.N return

while_loop:
        MOVS.N R3, #1                // r3 = 1
        ANDS.N R3, R3, R1            // R1 is even of oneven
        CMP.N R3, #1                //--> immediately lose function of R3
        BNE.N skip
        MOVS.N R3, R1
        MOVS.N R1, R0
        MOVS.N R0, R2
        MOVS.N R2, R3                // volgorde: R0 = p, R1 = n, R2 = m,
R3 = m
        PUSH.N {R1, R2, R3}          //--> push value of R1R2R3 on stack
        LDR.N R2, =multiply
        BLX.N R2
        POP.N {R1, R2, R3}
        MOVS.N R2, R0
        MOVS.N R0, R1
        MOVS.N R1, R3                // volgorde: R0 = n, R1 = m,
R2 = p, R3 = m
skip:
        PUSH.N {R1, R2}              // behoud waarde van m en p
        MOVS.N R1, R0                // R0 en R1 zijn n
        LDR.N R2, =multiply
        BLX.N R2
        POP.N {R1, R2}              // volgorde: R0 = n, R1 = m, R2 = p,
R3 = x
        LSRS.N R1, R1, #1

```

In C

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

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

```
extern int power(int a, int b);
extern int multiply(int a, int b);
```

```
int tothepower(int a, int b){
    int result = 1;
    for (int i = 0; i < b; i++){
        result = result * a;
    }
    return result;
}
```

```
int test_random_numbers(int number_of_tests){

    int succesful = 0;
    int a = 0;
    int b = 0;
    int result_assembly = 0;
    int result_c = 0;

    for (int i = 0; i < number_of_tests; i++){
        a = rand() % 5;
        b = rand() % 10;

        result_assembly = power(a, b);
        result_c = tothepower(a, b);

        if (result_assembly == result_c){
            succesful++;
        }
    }
    return succesful;
}
```

```
int test_limits(){
    int succesful = 0;
    int result_assembly = 0;
    int result_c = 0;

    result_assembly = power(0, 0);
    result_c = tothepower(0, 0);

    if (result_assembly == result_c){
        succesful++;
    }

    result_assembly = power(1, 0);
    result_c = tothepower(1, 0);

    if (result_assembly == result_c){
```


Resultaten

25 of the 25 tests have succeeded

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.

```
#define RCC_AHB1ENR_BIT_GPIODEN (*(volatile uint32_t*)(0x42000000 + 0x00023830 * 32 + 3 * 4))
#define GPIOD_BASE 0x40020C00
#define GPIOD_MODER (*(volatile uint32_t*)(GPIOD_BASE + 0x00))
#define GPIOD_ODR (*(volatile uint32_t*)(GPIOD_BASE + 0x14))
```

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.

```
11. GPIO->ODR = 0x5000;  
18. GPIO->ODR ^= 0xF000;
```

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

```
080001e4: ldr r3, [pc, #52] ; (0x800021c <main+68>)  
080001e6: mov.w r2, #1426063360 ; 0x55000000  
080001ea: str r2, [r3, #0]
```

2.2

```
080001e4: ldr r3, [pc, #52] ; (0x800021c <main+68>)  
080001e6: mov.w r2, #1426063360 ; 0x55000000  
080001ea: str r2, [r3, #0]
```

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 doet, 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


```
for (volatile int32_t i = 0; i < 100000; i++);  
LL_GPIO_TogglePin(GPIOD, (Green_LED_Pin | Red_LED_Pin));
```

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 bouwen 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:

>  .text	0x08000198	0x08000198	2,48 KB
---	------------	------------	---------

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

>  .text	0x08000198	0x08000198	1024 B
---	------------	------------	--------

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:


```
for (volatile int32_t i = 0; i < 1000000; i++);  
HAL_GPIO_TogglePin(GPIOD, Green_LED_Pin | Red_LED_Pin | Orange_LED_Pin |  
Blue_LED_Pin);
```

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

>  .text	0x08000198	0x08000198	5,17 KB
---	------------	------------	---------

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

>  .text	0x08000198	0x08000198	3,35 KB
--	------------	------------	---------

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_\(computer_programming\)](https://en.wikipedia.org/wiki/Volatile_(computer_programming))

Bijlage

OPDRACHT 1

```
#include <stdint.h>

#define RCC_AHB1ENR_BIT_GPIODEN *(volatile uint32_t*)(0x42000000 + 0x00023830 * 32 + 3 * 4)
#define GPIOD_BASE 0x40020C00
#define GPIOD_MODER *(volatile uint32_t*)(GPIOD_BASE + 0x00)
#define GPIOD_ODR *(volatile uint32_t*)(GPIOD_BASE + 0x14)
// There is something missing here ...

int main(void)
{
    // GPIO Port D Clock Enable
    RCC_AHB1ENR_BIT_GPIODEN = 1;
    // GPIO Port D Pin 15 down to 12 Push/Pull Output
    GPIOD_MODER = 0x55000000;
    // Set green and red LEDs
    GPIOD_ODR = 0x5000;
    // Do forever:
    while (1)
    {
        // Wait a moment
        for (volatile int32_t i = 0; i < 1000000; i++);
        // Flip all LEDs
        GPIOD_ODR ^= 0xF000;
    }
}
```

OPDRACHT 2

```
#include <stdint.h>
#include <stm32f4xx.h>

int main(void)
{
    // GPIO Port D Clock Enable
    RCC->AHB1ENR = RCC_AHB1ENR_GPIODEN;
    // GPIO Port D Pin 15 down to 12 Push/Pull Output
    GPIOD->MODER = GPIO_MODER_MODER12_0 | GPIO_MODER_MODER13_0 |
GPIO_MODER_MODER14_0 | GPIO_MODER_MODER15_0;
    // Set green and red LEDs
    GPIOD->ODR = 0x5000;
    /* There is something missing here ... = /* ... and here */
    // Do forever:
    while (1)
    {
        // Wait a moment
        for (volatile int32_t i = 0; i < 1000000; i++);
        // Flip all LEDs
        GPIOD->ODR ^= 0xF000;
        /* There is something missing here ... ^= /* ... and here */
    }
}
```


OPDRACHT 3

```
/* USER CODE BEGIN Header */
/**
 * *****
 * @file      : main.c
 * @brief     : Main program body
 * *****
 * @attention
 *
 * Copyright (c) 2022 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 * *****
 */
/* USER CODE END Header */
/* Includes ----- */
#include "main.h"

/* Private includes ----- */
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef ----- */
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ----- */
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro ----- */
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ----- */

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes ----- */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ----- */
/* USER CODE BEGIN 0 */
```

OPDRACHT 4

```
/* USER CODE BEGIN Header */
/**
 * *****
 * @file           : main.c
 * @brief          : Main program body
 * *****
 * @attention
 *
 * Copyright (c) 2022 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 * *****
 */
/* USER CODE END Header */
/* Includes ----- */
#include "main.h"

/* Private includes ----- */
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef ----- */
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ----- */
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro ----- */
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ----- */

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes ----- */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ----- */
/* USER CODE BEGIN 0 */
```