GEBZE TECHNICAL UNIVERSITY

DEPARTMENT OF

ELECTRONIC ENGINEERING

ELEC 458 - Embedded System Design

Project 1

REPORT

26/03/2019

Saim Buğrahan ÖZTÜRK | 151024011

Serhat SEFER | 141024040

Contents

Introduction

In this project,we implemented a fibonacci series calculator in ARM assembly.

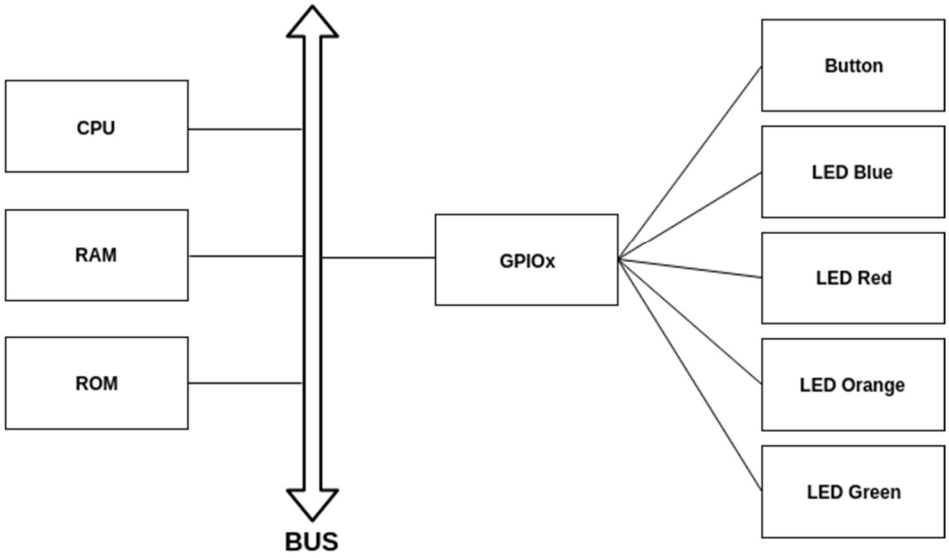
Our project goal was to develop an assembly code and burn it into Arm Cortex M4 processor in order to learn basics of the embedded systems.

In design algorithm process,we worked together due to being building blocks of the project.After the decision that, we drew the flowchart and blockdiagram according to algorithm.We were disassemble the code to functions.Serhat created the reading from button,displaying the group id and calculation of fibanacci functions.Buğra made displaying to results using binary encoding,delay functions and also wrote the report.

Block Diagram

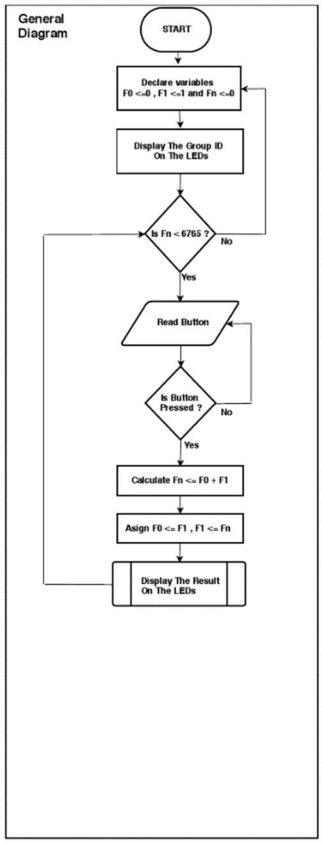
You can see below, the block diagram we used for our project.

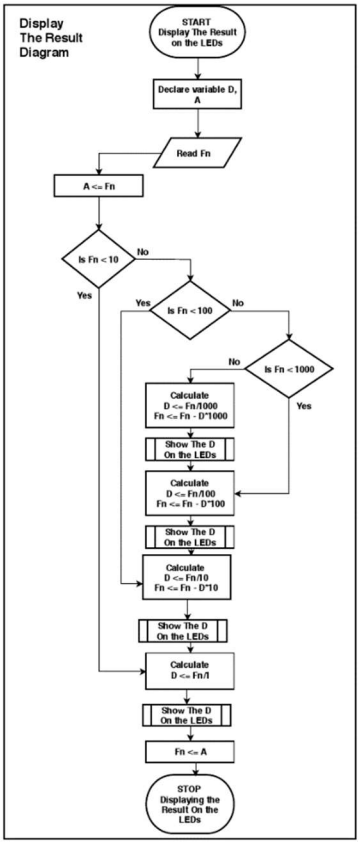
Saim Buğrahan ÖZTÜRK - 151024011
Serhat SEFER - 141024040



Software Flowchart

Below, you can see the flowchart we used to implement our project into assembly.We detailed the flowchart in order to help us later. Having this flowchart in our hand, we only had to think about how to implement it to assembly, which didn't take much time.





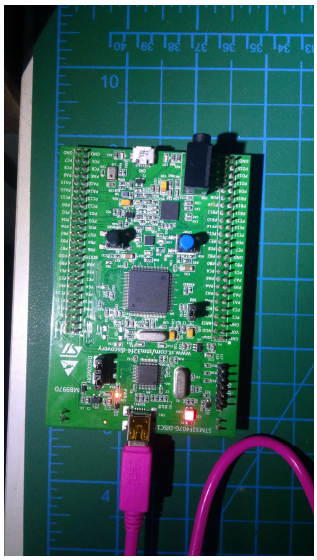
Design Overview

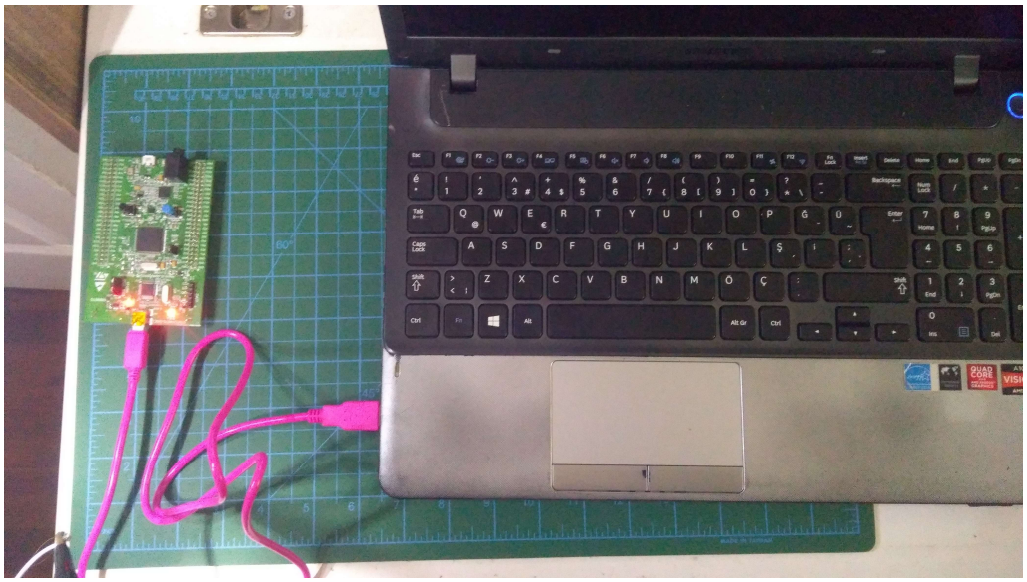
At first, designing process was a bit rough due to us not being involved in an assembly environment before. Thankfully, with the help of our instructer Mr. Furkan Çaycı, we were able to gain the insight about assembly coding which was required for our project.

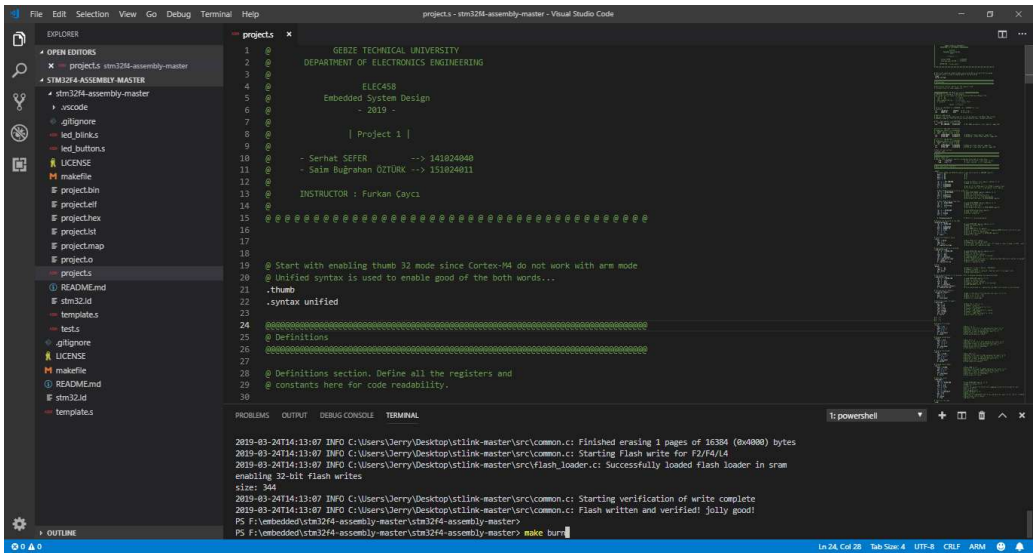
We chose the binary encoding algorithm to display the results.After decision,we designed the algorithm of project and drew the flowchart according to algorithm. The problem was how can we seperate the result to digits.After a little bit search we solved this problem.

Having our flowchart at hand, we managed to finish our assembly code within few hours. Challenging part was improving our code. After implementing our flowchart to assembly, we decided to use only one delay function in order to simplify and increase the readability of our code. To do so we had to use push{lr} and pop{pc} instructions which took our few more hours to figure out. Eventually we managed to finish our project in time with a well coded assembly program.

System Photo







Conclusion

It took around 15-20 hours to finish our project. By finishing the task, we gained a clean insight about what is going on inside the processor when we compile a C program. We gained the ability to read a product datasheet efficiently. We became familiar with GNU toolchain and we gained the necessary skills to debug a code and use a debugger.

As an another conclusion, we noticed that drawing a software flowchart before beginning the programming part really eases the task and that it should always be done in programming projects.

Grade Sheet

GEBZE TECHNICAL UNIVERSITY **DEPARTMENT OF ELECTRONIC ENGINEERING** **ELEC 458 - Embedded System Design** **Project 1** **STUDENT NAMES : Serhat Sefer** **: Saim Buğrahan ÖZTÜRK** **SIGN:**
PROJECT NAME: Project 1 – Fibonacci Series
PROJECT INSTRUCTERER: Dr. Furkan Çaycı **SIGN:**

Appendix

Assembly Code:

```
@             GEBZE TECHNICAL UNIVERSITY
@       DEPARTMENT OF ELECTRONICS ENGINEERING
@
@                   ELEC458
@           Embedded System Design
@                  - 2019 -
@
@                | Project 1 |
@
@      - Serhat SEFER         --> 141024040
@      - Saim Buğrahan ÖZTÜRK --> 151024011
@
@      INSTRUCTOR : Furkan Çaycı
@
@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @


@ Start with enabling thumb 32 mode since Cortex-M4 do not work with arm mode
@ Unified syntax is used to enable good of the both words...
.thumb
.syntax unified

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ Definitions
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ Definitions section. Define all the registers and
@ constants here for code readability.

@@@@@@@@@@@@@@@@@@ LED_DELAY Calculation @@@@@@@@@@@@@@@@@@
```

```
@ STM32F407 is running at 16 MHz Default.
@ Our delay function spend (Cortex-M4 Technical Reference Manual P.29)
@   sub r4 , #1             => 1 Cycle
@   cmp r4 , #0             => 1 Cycle
@   beq _read_button_low    => 1 or 1+pc
@   b _delay_0              => 1 + P Cycle (P=1)
@                             +__________
@                   TOTALY  => 6 Cycle
@
@ So we set LED_DELAY to (16000000 / 6) = 2666666 for 1 sec.
@ Constants
.equ     LED_DELAY,      2666666    @ For 1 sec
.equ     GROUP_ID,       0x01       @ Group ID = 1

@ Register Addressesm
@ You can find the base addresses for all the peripherals from Memory Map section
@ RM0090 on page 64. Then the offsets can be found on their relevant sections.

@ RCC   base address is 0x40023800
@   AHB1ENR register offset is 0x30
.equ     RCC_AHB1ENR,   0x40023830      @ RCC AHB1 peripheral clock register (page
180)

@*********** FOR LEDS ************
@ GPIOD base address is 0x40020C00
@   MODER register offset is 0x00
@   ODR   register offset is 0x14
.equ     GPIOD_MODER,   0x40020C00      @ GPIOD port mode register (page 281)
.equ     GPIOD_ODR,     0x40020C14      @ GPIOD port output data register (page
283)

@*********** FOR BUTTON ************
@ GPIOA base address is 0x40020000
```

```
@   MODER register offset is 0x00
@   IDR   register offset is 0x10
.equ     GPIOA_MODER,   0x40020000     @ GPIOA port mode register (page 281)
.equ     GPIOA_IDR,     0x40020010     @ GPIOA port input data register (page 283)

@ Start of text section
.section .text
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ Vectors
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ Vector table start
@ Add all other processor specific exceptions/interrupts in order here
    .long    __StackTop                 @ Top of the stack. from linker script
    .long    _start +1                  @ reset location, +1 for thumb mode

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ Main code starts from here
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

_start:
    @ Enable GPIOA and GPIOD Peripheral Clock (bit 0 and 3 in AHB1ENR register)
    movs r1, #0                         @ F0
    movs r2, #1                         @ F1
    movs r3, #0                         @ Fn
    movs r8, #0                         @ D

    ldr r6, = RCC_AHB1ENR               @ Load peripheral clock register address to r6
    ldr r5, [r6]                        @ Read its content to r5
    and r5, 0x00000000
```

```
    orr r5, 0x00000009                  @ Set bit 0 for GPIOA and 3 for GPIOD to
enable clock

    str r5, [r6]                        @ Store back the result in peripheral
clock register


    @ Make GPIOD Pin12,13,14,15 as output pin

    ldr r6, = GPIOD_MODER               @ Load GPIOD MODER register address to r6

    ldr r5, [r6]                        @ Read its content to r5

    and r5, 0x00FFFFFF                  @ Clear bits for P12,P13,P14,P15

    orr r5, 0x55000000                  @ Write 01 to bits for P12,P13,P14,P15

    str r5, [r6]                        @ Store back the result in GPIOD MODER
register


    @ Make GPIOA Pin0 as input pin

    ldr r6,= GPIOA_MODER                @ Load GPIOA MODER register address to r6

    ldr r5 , [r6]                       @ Read its content to r5

    and r5,  0xFFFFFF00                 @ Write 00 to bits P0

    str r5 , [r6]                       @ Store back the result in GPIOA MODER
register


    ldr r6, = GPIOD_ODR                 @ Load GPIOD output data register

    ldr r5, [r6]                        @ Read its content to r5

    and r5, 0x0000                      @ write 1 to pin 12

    str r5, [r6]


    b _display_group_id                 @ Branch to _display_group_id


@ Display the Group ID on the LEDs

_display_group_id:

    ldr r6,= GPIOD_ODR                  @ Load GPIOD_ODR register address to r6

    ldr r5, [r6]                        @ Read its content to r5

    and r5, 0x00000000                  @ Clear R5

    mov r0, #GROUP_ID                   @ Move GROUP_ID to r0 register

    lsl r0,r0,#12                       @ Shift r0 register to 12 bit left
(mapping GROUP_ID with 12,13,14,15. pin)
```

```
    orr r5, r0                          @ Write r0 to r5 (or operation)

    str r5, [r6]                        @ Store back the result in GPIOD_ODR
register

    b _check                            @ Branch to _check


@ Check the Fibonacci value

_check:

    mov r4,#6765                        @ Move 6765 to r4 register

    cmp r3,r4                           @ Compare R3 (Fn) and R4 (6765)

    beq _start                          @ If is equal then branch to _start (If
fibonacci value is equal to 6765, reset program.)

    b _read_button_high                 @ Else branch to _read_button_high


@ Check whatif button is pressed

_read_button_high:

    ldr r6 , = GPIOA_IDR                @ Load GPIOA_IDR register address to r6

    ldr r5 , [r6]                       @ Read its content to r5

    and r5 , #0x1                       @ Catch r5's 0. bit (a0 pin)

    cmp r5 , #0x1                       @ Compare r5 and 1 (button is pressed)

    ldr r6,=LED_DELAY                   @ Load LED_DELAY to r4

    bne _read_button_high               @ If not equal 0 branch back to
_read_button_high (Wait until button is pressed)

    bl  _delay                          @ Branch and link to _delay

    b _read_button_low                  @ Branch _read_button_low


@ Delay

_delay:

    sub r6 , #1                         @ Subtract 1 from r4 register (LED_DELAY)

    cmp r6 , #0                         @ Compare r4 and 0

    bne _delay                          @ Else branch back to _delay_0  (Subtract
until r4 is equal to 0)

    bx  lr                              @ Branch and execute lr


@ Check whatif button is not pressed. (This is because avoiding continously
pressed)
```

```
_read_button_low:
    ldr r6 , = GPIOA_IDR                @ Load GPIOA_IDR register to r6
    ldr r5 , [r6]                       @ Read its content to r5
    and r5 , #0x1                       @ Catch r5's 0. bit (a0 pin)
    cmp r5 , #0x0                       @ Compare r5 and 0 (button is not pressed)
    ldr r4,=LED_DELAY                   @ Load LED_DELAY to r4
    beq _calculate_fibonacci            @
    b _read_button_low                  @ Else branch back to _read_button_low
(Wait until button is not pressed)


@ Calculation of fibonacci
_calculate_fibonacci:
    add r3,r1,r2                        @ Add r1 (F0) and r2 (F1) and move the
result to r3 (Fn)
    movs r1,r2                          @ Move r2 to r1 (F1 => F0)
    movs r2,r3                          @ Move r3 to r2 (Fn => F1)
    b _check_digit                      @ Branch to _check_digit


@ Check the number of digits
_check_digit:
    mov r4, r3                          @ Move the r3 (Fn) to r4
    cmp r4,#10                          @ Compare r4 and 10
    bls _first                          @ If r4 <= 10 then branch to _first
    cmp r4,#100                         @ Compare r4 and 100
    bls _second                         @ If r4 <= 100 then branch to _second
    cmp r4,#1000                        @ Compare r4 and 1000
    bls _third                          @ If r4 <= 1000 then branch to _third
    b _fourth                           @ Else branch to _fourth


@r8 -> D
@r4 -> A
@r6 -> Fn
```

```
@ Display first digit

_first:

    movs r5,#1                      @ Move 1 to r5

    udiv r8,r4,r5                   @ Divide r4 (Fn) to r5 (1) and move the
result to r8

    mul r6,r8,r5                    @ Multiply r8 and r5 (1) and move the
result to r6

    sub r4,r6                       @ Subtract r4 and r6 and move the result to
r4

    bl _show_digit                  @ Branch and link to _show_digit

    b _check                        @ Then branch back to _check


@ Display second digit

_second:

    movs r5,#10                     @ Move 10 to r5

    udiv r8,r4,r5                   @ Divide r4 (Fn) to r5 (10) and move the
result to r8

    mul  r6,r8,r5                   @ Multiply r8 and r5 (10) and move the
result to r6

    sub  r4,r6                      @ Subtract r4 and r6 and move the result to
r4

    bl _show_digit                  @ Branch and link to _show_digit

    b _first                        @ Then branch to _first


@ Display third digit

_third:

    movs r5,#100                    @ Move 100 to r5

    udiv r8, r4,r5                  @ Divide r4 (Fn) and r5 (100) and move the
result to r8

    mul r6,r8,r5                    @ Multiply r8 and r5 (100) and move the
result to r6

    sub r4,r6                       @ Subtract r4 and r6 and move the result to
r4

    bl _show_digit                  @ Branch and link to _show_digit

    b _second                       @ Then branch to _second
```

```
@ Display fourth digit

_fourth:

    movs r5,#1000                       @ Move 1000 to r5

    udiv r8, r4,r5                      @ Divide r4 (Fn) and r5 (1000) and move the
result to r8

    mul r6,r8,r5                        @ Multiply r8 and r5 (1000) and move the
result to r6

    sub r4,r6                           @ Subtract r4 and r6 and move the result to
r4

    bl _show_digit                      @ Branch and link to _show_digit

    b _third                            @ Then branch to _third


@ Show the result

_show_digit:

    push {lr}

    ldr r6,= GPIOD_ODR                  @ Load GPIOD_ODR register to r6

    ldr r5, [r6]                        @ Read its content to r5

    and r5, 0x0                         @ Clear r5

    lsl r8,r8,#12                       @ Shift r8 register to 12 bit left (mapping
r8 with 12,13,14 and 15. pin)

    orr r5, r8                          @ Write r8 to r5

    str r5, [r6]                        @ Store back to r6

    ldr r6,=LED_DELAY                   @ Load LED_DELAY to r6

    mov r5,#2                           @ Move r5 to 2

    mul r6,r5                           @ Multiply r6 (LED_DELAY) and r5 (2) and
move the result to r6. (So r6 = 2*LED_DELAY = 2 sec)

    bl _delay                           @ Branch and link to _delay

    b _stop                             @ Branch _stop


@ Light off the LEDs

_stop:

    ldr r6,= GPIOD_ODR                  @ Load GPIOD_ODR register to r6

    ldr r5, [r6]                        @ Read its content to r5

    and r5, 0x0                         @ Clear r5

    str r5, [r6]                        @ Store back to r6
```

```
    ldr r6,=LED_DELAY                   @ Load LED_DELAY to r6

    mov r5,#3                           @ Move r5 to 3

    mul r6,r5                           @ Multiply r6 (LED_DELAY) and r5 (3) and
move result to r6. (So r6 = 3*LED_DELAY = 3 sec)

    bl _delay                           @ Branch _delay_3

    pop {pc}                            @ Pop pc
```