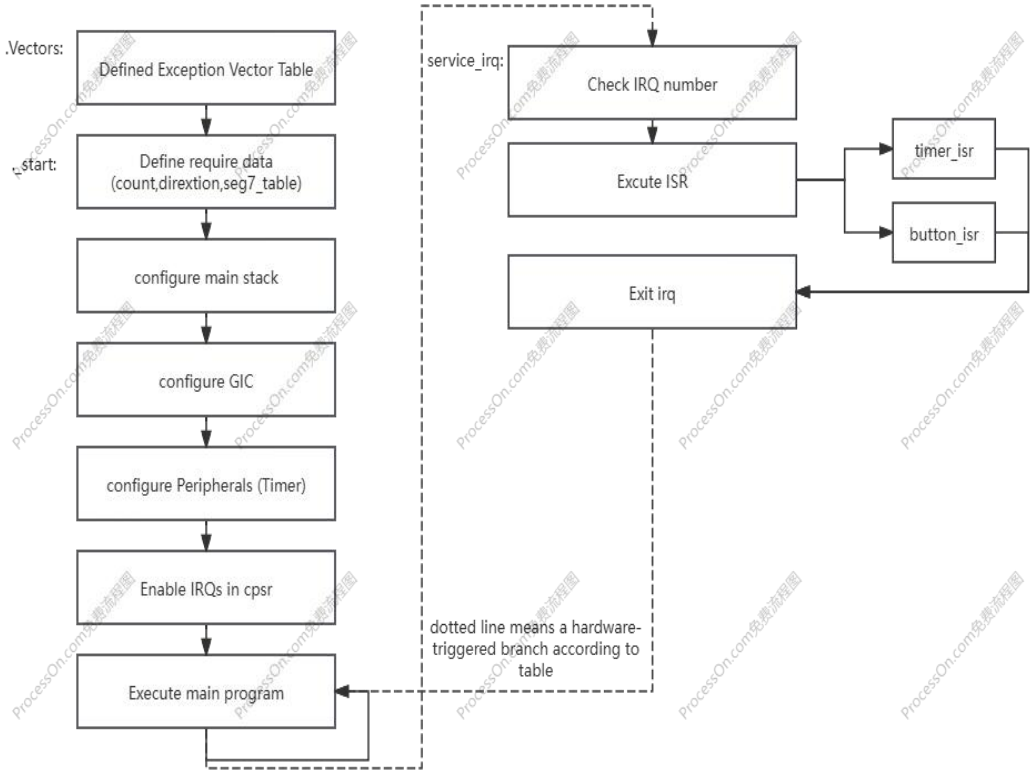
Lab2 Report

Shihao luo

Statement

Create a three-digit decimal counter that accurately counts 1 s intervals. The count should be displayed as a decimal number on the 7-segment display. An interrupt-enabled push button should be used to reverse the direction of the counter. The count should roll over to zero after reaching 999 (if counting up), or vice versa if counting down.

Explanation



Approached and implemented the problem

In this project, I followed a structured approach to implement the required functionality in four main steps: defining the required data and exception vector table, configuring the initial step for the interrupt, setting the service_irq function, and completing the implementation of the timer_isr and button_isr functions.

Step 1: Defining the required data and exception vector table
The first step was to define the required data and the exception vector table. For the required data, count is used to calculate the real-time digit, direction is used to change the counting direction, and seg7_table is used to configure the seven-segment display. Regarding the exception vector data, since this program uses the service_irq, the other vectors are kept null.

Step 2: configuring the initial step for the interrupt
Next, since we need to execute service_irq latter, it is essential to initialize the interrupt.There are a few steps to realize it, they are:Configure IRQ stack, Configure main stack, Configure GIC, Configure Peripherals, Enable IRQs in cpsr and Execute the main program, which is a dead loop.

Step 3: Setting the service_irq function
There are three steps to execute service_irq, they are: Check the irq number, excute irq and write end of the interrupt.

Step 4: completing the implementation of the timer_isr and button_isr functions.
The final step was to complete the implementation of timer_isr and button_isr. For the timer_isr, what we need to realize is to make a judgement on the value of 999 and 0, cause over and below these values will lead to overflow and we need to avoid those situation, then is to convert the digits into seven-segment display. For the button_isr part, i define a variant ‘direction’ to control the direction of counting numbers.

Debugging and Challenges
One of the main challenges I faced was that i do not know how to sequentially convert the digits to the seven-segment display. I spend lots of time searching the information on the internet and looking up the related books. And the other challenges is that i always forget to perverse register when i implement the new function.

In conclusion, by breaking down the project into smaller, testable components, I was able to systematically implement and verify each part. This approach helped me maintain clarity throughout the development process and effectively troubleshoot any issues that arose.

TIME CONSUMPTION AND FUTHER INPROVEMENT

For this project, I spent the first hour reviewing the relevant concepts, as I had forgotten most of the details about timers and the seven-segment display. During this time, I also analyzed my approach and outlined my solution. Initially, I planned

to complete the coding within an hour. However, I encountered challenges with equentially convert the digits to the seven-segment display , which extended the total development time to three hours.

Although I successfully completed the project, I recognize a significant limitation in my current implementation. As it stands, I will try to display four decimal digits on the seven-segment display.

I believe it would be worthwhile to improve this aspect of my code. Developing a more dynamic and scalable solution would not only make my code more efficient but also deepen my understanding of assembly language and hardware interactions. Investing time in this enhancement would ultimately benefit my long-term learning and coding practices.

Code

```
.section .vectors, "ax", %progbits

.align 4

    b _start

    b evt_undef

    b evt_undef

    b evt_undef

    b evt_undef

    .word 0

    b service_irq

    b evt_undef


.data

.align 4
```

```
count:          .word    0
direction:    .word    1

seg7_table:
.word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F

.text
.global _start
_start:
    mov r1, #0b11010010
    msr cpsr_c, r1
    ldr sp, =0xFFFFFFFC

    mov r1, #0b11010011
    msr cpsr, r1
    ldr sp, =0x3FFFFFFC

    bl config_gic

    ldr r0, =0xFF202000
    mov r1, #8
    str r1, [r0, #4]
    str r1, [r0]
    ldr r1, =100000000
```

```
    str r1, [r0, #8]
    lsr r1, #16
    str r1, [r0, #12]
    mov r1, #7
    str r1, [r0, #4]

    ldr r0, =0xFF200050
    mov r1, #1
    str r1, [r0, #8]

    mov r0, #0b01010011
    msr cpsr_c, r0

idle:
    b idle

evt_undef:
    b evt_undef

service_irq:
    push {r0-r7, r12, lr}

    ldr r4, =0xFFFEC100
    ldr r5, [r4, #12]
```

```
    cmp r5, #72
    beq timer_isr

    cmp r5, #73
    beq button_isr

exit_irq:
    str r5, [r4, #16]
    pop {r0-r7, r12, lr}
    subs pc, lr, #4

timer_isr:
    ldr r0, =0xFF202000
    mov r1, #1
    str r1, [r0]

    ldr r0, =count
    ldr r1, [r0]

    ldr r3, =direction
    ldr r3, [r3]
    adds r1, r1, r3
```

```
    ldr r2, =0
    cmp r1, r2
    blt set_to_max

    ldr r2, =999
    cmp r1, r2
    ble valid_count
    ldr r1, =0
    b store_count

set_to_max:
    ldr r1, =999

valid_count:
store_count:
    str r1, [r0]
    bl display_count
    b exit_irq

display_count:
    push {lr}

    mov r12, r1
    mov r0, r12
```

```
ldr r1, =10

bl __modsi3

mov r2, r0


mov r0, r12

ldr r1, =10

bl __divsi3

mov r12, r0


mov r0, r12

ldr r1, =10

bl __modsi3

mov r3, r0


mov r0, r12

ldr r1, =10

bl __divsi3

mov r1, r0


ldr r0, =seg7_table

lsl r12, r2, #2

ldr r2, [r0, r12]

lsl r12, r3, #2

ldr r3, [r0, r12]
```

```
    lsl r12, r1, #2
    ldr r1, [r0, r12]

    lsl r3, r3, #8
    orr r2, r2, r3
    lsl r1, r1, #16
    orr r2, r2, r1

    ldr r0, =0xFF200020
    str r2, [r0]

    pop {lr}
    bx lr

button_isr:
    ldr r0, =0xFF200050
    mov r1, #1
    str r1, [r0, #12]

    ldr r0, =direction
    ldr r1, [r0]
    rsb r1, r1, #0
    str r1, [r0]
    b exit_irq
```

```
.global config_gic
config_gic:
    push {lr}

    mov r0, #72
    mov r1, #1
    bl config_interrupt

    mov r0, #73
    mov r1, #1
    bl config_interrupt

    ldr r0, =0xFFFEC100
    ldr r1, =0xFFFF
    str r1, [r0, #4]
    mov r1, #1
    str r1, [r0]

    ldr r0, =0xFFFED000
    str r1, [r0]

    pop {lr}
    bx lr
```

```
config_interrupt:
    push {r4-r5, lr}

    lsr r4, r0, #5
    ldr r2, =0xFFFED100
    add r4, r2, r4, LSL #2
    and r2, r0, #0x1F
    mov r5, #1
    lsl r2, r5, r2
    ldr r3, [r4]
    orr r3, r3, r2
    str r3, [r4]

    ldr r2, =0xFFFED800
    add r4, r2, r0
    strb r1, [r4]

    pop {r4-r5, lr}
    bx lr

__divsi3:
    push {r2, r3, lr}
    mov r2, #0
```

```
div_loop:
    cmp r0, r1
    blt div_end
    subs r0, r0, r1
    adds r2, r2, #1
    b div_loop
div_end:
    mov r0, r2
    pop {r2, r3, lr}
    bx lr

__modsi3:
    push {r2, r3, lr}
    mov r2, r0
    bl __divsi3
    mul r3, r0, r1
    subs r0, r2, r3
    pop {r2, r3, lr}
    bx lr

.end
```