GitHub Link

https://github.com/LuthoYRN/MNGLUT008_CBXLIS001_EEE3096S/blob/main/Prac4/assembly.s

Description of Implementation

Delay subroutines

Long Delay Subroutine (long_delay): This subroutine is used to create a delay of 0.7 seconds. It does this by loading the value 1400000 which is declared as LONG_DELAY_CNT into a register and then decrementing the register's value until it reaches zero. Each iteration of this loop represents a unit of delay, and once the register value reaches zero, the delay completes and branches back using link register, allowing the program to proceed.

Short Delay Subroutine (short_delay): The short delay subroutine generates a delay of 0.3 seconds. Like the long delay, it initializes a register with a smaller value 600000 loaded from SHORT_DELAY_CNT to achieve a faster interval. The loop structure here works identically as long_delay.

Main loop operation

Firstly, the value of the IDR register is read to gather the current states of the push buttons. The loop continuously monitors the state of the pushbuttons and determines which action to take based on their input states. This is done using a series of condition checks (using bitmask operations) to identify which buttons are pressed, true condition checks trigger branches to various subroutines.

1. **Default LED Increment**
   When no buttons are held down meaning none of the bitmasks trigger a branch, the program enters the default behaviour state where the LEDs increment by 1 every 0.7 seconds. It does this by branching to the no_button_pressed subroutine which just increments the ODR register by 1 and calls write_leds which stores the new ODR value and calls long_delay for 0.7 second delay.
2. **SW0 and SW1 Pressed**
   If SW0 and SW1 are simultaneously held down, the program branches to sw0_sw1_pressed subroutine which increments the ODR register by 2 every 0.3 seconds utilising the short_delay subroutine to maintain the interval.
3. **SW0 Pressed**
   If SW0 is held down, the program branches to sw0_pressed subroutine which increments the ODR register by 2 every 0.7 seconds utilising the long_delay subroutine to maintain the interval.
4. **SW1 Pressed**
   If SW1 is held down, the program branches to sw1_pressed subroutine which triggers a faster increment rate, updating the ODR register by 1 every 0.3 seconds by invoking the short_delay subroutine.
5. **SW2 Pressed**
   If SW2 is pressed, the program branches to sw2_pressed subroutine where the LED pattern immediately changes to a fixed value of OxAA done by overwriting the ODR register with the value.
6. **SW3 Pressed**
   The program branches to sw3_pressed subroutine and enters a freeze state where the LED pattern remains unchanged, the subroutine does this by just branching back to the main loop every time and not doing anything to the ODR register. This effectively freezes the display until SW3 is released.

```
/*
 * assembly.s
 *
 */

 @ DO NOT EDIT
      .syntax unified
    .text
    .global ASM_Main
    .thumb_func

@ DO NOT EDIT
vectors:
      .word 0x20002000
      .word ASM_Main + 1

@ DO NOT EDIT label ASM_Main
ASM_Main:

      @ Some code is given below for you to start with
      LDR R0, RCC_BASE             @ Enable clock for GPIOA and B by setting bit 17
and 18 in RCC_AHBENR
      LDR R1, [R0, #0x14]
      LDR R2, AHBENR_GPIOAB   @ AHBENR_GPIOAB is defined under LITERALS at the end of
the code
      ORRS R1, R1, R2
      STR R1, [R0, #0x14]

      LDR R0, GPIOA_BASE           @ Enable pull-up resistors for pushbuttons
      MOVS R1, #0b01010101
      STR R1, [R0, #0x0C]
      LDR R1, GPIOB_BASE      @ Set pins connected to LEDs to outputs
      LDR R2, MODER_OUTPUT
      STR R2, [R1, #0]
      MOVS R2, #0             @ NOTE: R2 will be dedicated to holding the value on the
LEDs
      MOVS R3, #0

@ TODO: Add code, labels and logic for button checks and LED patterns

main_loop:
    LDR R5, GPIOA_BASE
    LDR R3, [R5, #0x10]       @ Reading input data register (IDR)

    MOVS R5, #0b00000011      @ Set R5 to 0b00000010 (mask for bit 1 - SW0)
    TST R3, R5                @ Test bit 1 by ANDing R3 and R5; sets condition flags
    BEQ sw0_sw1_pressed       @ If Z flag is set, SW0 is pressed

    MOVS R5, #0b00000001      @ Set R5 to 0b00000010 (mask for bit 1 - SW0)
    TST R3, R5                @ Test bit 1 by ANDing R3 and R5; sets condition flags
    BEQ sw0_pressed           @ If Z flag is set, SW0 is pressed

    MOVS R5, #0b00000010      @ Set R5 to 0b00000010 (mask for bit 1 - SW0)
    TST R3, R5                @ Test bit 1 by ANDing R3 and R5; sets condition flags
    BEQ sw1_pressed           @ If Z flag is set, SW0 is pressed

    MOVS R5, #0b00000100      @ Set R5 to 0b00000010 (mask for bit 1 - SW0)
    TST R3, R5                @ Test bit 1 by ANDing R3 and R5; sets condition flags
    BEQ sw2_pressed           @ If Z flag is set, SW0 is pressed

    MOVS R5, #0b00001000      @ Set R5 to 0b00000010 (mask for bit 1 - SW0)
    TST R3, R5                @ Test bit 1 by ANDing R3 and R5; sets condition flags
    BEQ sw3_pressed           @ If Z flag is set, SW0 is pressed

    B no_button_pressed
```

```
    B main_loop              @ Loop back to the start if no match


no_button_pressed:
    @ Default behavior: increment LEDs by 1 every 0.7 seconds
    BL long_delay            @ Call long delay function
    ADDS R2, R2, #1          @ Increment LEDs by 1
    B write_leds             @ Go write the LED values

sw0_pressed:
    @ SW0 behavior: increment LEDs by 2
    BL long_delay            @ Call long delay function
    ADDS R2, R2, #2          @ Increment LEDs by 2
    B write_leds             @ Go write the LED values

sw1_pressed:
    @ SW1 behavior: faster increment (0.3 seconds)
    BL short_delay           @ Call short delay function
    ADDS R2, R2, #1          @ Increment LEDs by 1
    B write_leds             @ Go write the LED values

sw2_pressed:
    @ SW2 behavior: set LED pattern to 0xAA
    MOVS R2, #0xAA           @ Set LEDs to pattern 0xAA
    B write_leds             @ Go write the LED values

sw0_sw1_pressed:
    @ If both SW0 and SW1 are pressed, increment by 2 every 0.3 seconds
    ADDS R2, R2, #2          @ Increment LEDs by 2
    BL short_delay           @ Call short delay function (0.3 seconds)
    B write_leds             @ Go write the LED values

sw3_pressed:
    @ SW3 behavior: freeze the LEDs
    B main_loop              @ Just loop without updating LEDs

////Delay methods
long_delay:
    LDR R4, LONG_DELAY_CNT   @ Load long delay count into R4
    B delay_loop

@ Subroutine for short delay (0.3 seconds)
short_delay:
    LDR R4, SHORT_DELAY_CNT  @ Load short delay count into R4
    B delay_loop             @ Use the same loop as in long_delay

delay_loop:
    SUBS R4, R4, #1          @ Decrement R4 (not R0)
    BNE delay_loop           @ Loop until R4 reaches zero
    BX LR
                  @ Return from subroutine
write_leds:
    STR R2, [R1, #0x14]
    B main_loop

@ LITERALS; DO NOT EDIT
    .align
RCC_BASE:           .word 0x40021000
AHBENR_GPIOAB:      .word 0b11000000000000000000
GPIOA_BASE:         .word 0x48000000
GPIOB_BASE:         .word 0x48000400
MODER_OUTPUT:       .word 0x5555

@ TODO: Add your own values for these delays
LONG_DELAY_CNT:     .word 1400000    @ 0.7 second delay
SHORT_DELAY_CNT:    .word 600000     @ 0.3 second delay
```