EE306 - Microprocessors

Laboratory Exercise 5
Input/Output in an Embedded System

May 1, 2020

Alp Gokcek
#041701014

1 Part I - Blink LED in 0.25 second Intervals

In this experiment, I've used polled I/O to make the processor wait for the timer for 0.25 second intervals. Since our processor's frequency is 200 MHz, to make it wait for 0.25 seconds, we need to solve the equation $(1/200MHz) * x = 0.25s$ for x. From the equation, we find x to be equal to 50000000. We load this item and set the control bits and jump to main loop.
Important Note: While running this assembly code on CPUlator, "Function clobbered callee-saved register" interrupt must be disabled.

Listing 1: Assembly Code for blinking LEDs in 0.25 seconds

```
.global _start
 _start:
        LDR R0, =0xFF200000 // base address of LEDs
        LDR R1, =0xFFFEC600 // R1 points to timer.
        LDR R2, =50000000  // R2 is control word.
        STR R2, [R1] // update timer load register
        MOV R2, #0b011 // set control bits to I->0, A->1, E->1
        STR R2, [R1, #8] //update timer control register
        MOV R2, #1 // blinking flag

 loop: STR R2, [R0] // led is on.
        waitloop: LDR R4, [R1, #0xC] // status reg.
        CMP R4, #1 // if 1, 0.25 second is completed.
        BNE waitloop
        STR R4, [R1, #0xC] // reset status flag.
        EOR R2, R2, #1 // update blinking flag
 B loop

 END: B END
 .end
```

2 Part II - Real-Time Clock

In this experiment, I've converted each decimal number from 0-9 into their 7-segment equivalent and stored in memory. In each 0.01 second interval, I have incremented the number displayed on the 7-segment display by 1. There are four pointers for each HEX_{3-0} display, and it gets updated while in each 0.01 second time interval. We also have several push-buttons, and if at least one of them is pressed, until the time that all of the buttons get on the unpressed state, the timer stops.
Important Note: While running this assembly code on CPUlator, "Function clobbered callee-saved register" interrupt must be disabled.

Listing 2: Assembly code for real-time clock

```
.global _start
_start:
       LDR R0, DISPLAY_BASE // base address of 7-segment displays
       LDR R1, TIMER_BASE // R1 points to timer.
       LDR R2, =2000000 // R2 is control word for 0.01s
       STR R2, [R1] // update timer load register
       MOV R2, #0b011 // set control bits to I->0, A->1, E->1
       STR R2, [R1, #8] //update timer control register
       LDR R2, =DECIMAL_TO_7_SEG // decimal to 7 segment memory location pointer
       MOV R4, #0 // hex 3 memory location pointer
       MOV R5, #0 // hex 2 memory location pointer
```

```
        MOV R6, #0 // hex 1 memory location pointer
        MOV R7, #0 // hex 0 memory location pointer
loop:
        BL check_is_stopped
        waitloop: LDR R3, [R1, #0xC] // status reg.
        CMP R3, #1 // if 1, 0.01 second is completed.
        BNE waitloop
        STR R3, [R1, #0xC] // reset status flag.
        BL update_hex0_val // update hex0
        BL update_display // update display
B loop

update_hex0_val:
        PUSH {LR} // save link register
        ADD R7, R7, #1 // increment pointer
        CMP R7, #10 // check if it is 10
        MOVEQ R7, #0 // if it is 10, make it 0
        BLEQ update_hex1_val // update hex1 if r7 was 10
        POP {LR} // restore link register
        BX LR // end subroutine

update_hex1_val:
        PUSH {LR} // save link register
        ADD R6, R6, #1 // increment pointer
        CMP R6, #10 // check if it is 10
        MOVEQ R6, #0 // if it is 10, make it 0
        BLEQ update_hex2_val // update hex2 if r6 was 10
        POP {LR} // restore link register
        BX LR // end subroutine

update_hex2_val:
        PUSH {LR} // save link register
        ADD R5, R5, #1 // increment pointer
        CMP R5, #10 // check if it is 10
        MOVEQ R5, #0 // if it is 10, make it 0
        BLEQ update_hex3_val // update hex3 if r5 was 10
        POP {LR} // restore link register
        BX LR // end subroutine


update_hex3_val:
        ADD R4, R4, #1 // increment pointer
        CMP R4, #6 // check if it is 6
        MOVEQ R4, #0 // if it is 6, make it 0
        BX LR // end subroutine

update_display:
        PUSH {R2, R4, R5, R6, R7} // save registers
        LDR R4, [R2, R4, LSL #2] // hex3 value
        LDR R5, [R2, R5, LSL #2] // hex2 value
        LDR R6, [R2, R6, LSL #2] // hex1 value
        LDR R7, [R2, R7, LSL #2] // hex0 value
        MOV R2, #0
        ADD R2, R2, R4, LSL #24 // hex3
```

```
        ADD R2, R2, R5, LSL #16 // hex2
        ADD R2, R2, R6, LSL #8 // hex1
        ADD R2, R2, R7 // hex0
        STR R2, [R0] // update display
        POP {R2, R4, R5, R6, R7} // restore registers
        BX LR

check_is_stopped:
        PUSH {LR, R0, R1} // save register contents
        LDR R0, PUSH_BUTTON_BASE // base address of push buttons
        BL is_stopped // branch to is_stopped subroutine
        POP {LR, R0, R1} // restore register contents
        BX LR // return to main loop
is_stopped:
        LDR R1, [R0] // load push buttons state
        CMP R1, #0 // check if any of them is pressed
        BNE is_stopped // if any of them is pressed branch to is_stopped
        BX LR // if none of them is pressed, return to check_is_stopped

DISPLAY_BASE: .word 0xff200020 // display base address
TIMER_BASE: .word 0xfffec600 // timer base address
PUSH_BUTTON_BASE: .word 0xff200050 // push button base address

// converted versions of numbers 0-9 to display on 7-segment display
DECIMAL_TO_7_SEG: .word 63,6,91,79,102,109,125,7,127,111
END: B END
.end
```