Experiment 6

Sevim Eftal Akşehirli, 150190028
Hacer Yeter Akıncı, 150200006
Büşra Özdemir, 150200036
Aslı Yel, 150200054

***Abstract*—This experiment explores the crucial role of timers in microprocessors through various sections. It begins by lighting different digits on a 7-segment display simultaneously and then proceeds to create a subroutine converting binary inputs into decimal values for the display. The focus then shifts to configuring and using TimerA in MSP430 microcontrollers, specifically in compare mode with up counting, with details on setting up 10-millisecond interval interrupts. Leveraging these interrupts, a chronometer is built to display centiseconds and seconds on the 7-segment display, controlled by three buttons for pausing, starting, and resetting. Throughout, this experiment highlights practical timer applications, interrupts, and user-controlled functions within embedded systems.**

***Index Terms*—TimerA, Timer Interrupt, Control Signal, SMCLK**

I. Introduction

This experiment focuses on understanding timers in microprocessors, highlighting their importance in managing tasks. It involves lighting segments on a 7-segment display together, creating a binary-to-decimal conversion subroutine, and using timer interrupts to build a chronometer. These steps show how timers are crucial for controlling time-related functions in embedded systems.

II. Materials and Methods

Listing 1: Assembly Code for Part1

```
Setup       mov.b   #11111111b, &P1DIR
            mov.b   #00000000b, &P1OUT
            mov.b   #00001111b, &P2DIR

Start       mov.w   #numbers, R4
            mov.b   #00000001b, R5

Mainloop    clr     P1OUT
            clr     P2OUT
            mov.b   0(R4), P1OUT
            mov.b   R5, P2OUT
            add     #1d, R4
            cmp     #endOfNumbers, R4
            jeq     Start
            rla     R5
            jmp     Mainloop

            .data
numbers     .byte   00111111b, 00000110b,
                    01011011b, 01001111b
endOfNumbers
```

PART1:

Setup: This section sets the P1 port as output, and sets the P2 port least significant 4 bits as output.

Start: This section loads the R4 with the address of the 'numbers' array, and R5 is initialized with the value 00000001b.

Mainloop: This loop clears the output ports, loads the value at the memory address in R4 into P1OUT, moves the value in R5 into P2OUT, increments the array index, and compares it with the end address, if equal it goes back to Start section, otherwise, rotates R5 to the left, and goes back to mainloop.

Listing 2: Assembly Code for Part2

```
Setup       bis.b   #11111111b, &P1DIR
            bis.b   #00000000b, &P1OUT
            bis.b   #00001111b, &P2DIR
            mov.b   #49d, R4
            mov.b   #0d, R5
            mov.b   #00001000b, R6

Convertion  cmp     #00001010b, R4
            jl      Display1
            sub.b   #00001010b, R4
            inc.b   R5
            jmp     Convertion

Display1    clr     P1OUT
            clr     P2OUT
            xor.b   #00001100b, R6
            mov.w   #numbers, R7
            add     R5, R7
            mov.b   0(R7), P1OUT
            mov.b   R6, P2OUT

Display2    clr  P1OUT
            clr  P2OUT
            xor.b #00001100b, R6
            mov.w #numbers, R7
            add  R4, R7
            mov.b 0(R7), P1OUT
            mov.b R6, P2OUT
            jmp  Display1
```

```
            .data
numbers     .byte 00111111b, 00000110b,
01011011b, 01001111b, 01100110b, 01101101b,
01111101b, 00000111b, 01111111b, 01101111b

endOfNumbers
```

PART2:

Setup: Input/Output ports are configured according to port connections of 7 segment display. Port1 is set as output which will decide which character will be displayed on the seven-segment. Bits 3-0 of Port2 are set as output and will be used to select which 7-segment display will currently display a character. Also the decimal number to be displayed is put into R4. Also R5 is initially set to 0 and will be used to store the tens value of the given number. R6 is set so 00001000b. When is will be passed to P2OUT, the rightmost seven-segment display will be selected.

Convertion: This section subtracts 10 from R4 and overwrites the result to R4 until R4 is less than 10. After every subtraction R5 is increased by 1, indicating that number has one more 10 in it. At the end R5 keeps the tens value, R4 keeps the ones value of the given number.

Display1: P1OUT and P2OUT is cleared for a clear transition between selected seven-segments. R6 is modified to 00000100b and the second rightmost seven-segment will display. What it displays is the tens value of the given number.

Display2: P1OUT and P2OUT is cleared for a clear transition between selected seven-segments. R6 is modified to 00001000b and the rightmost seven-segment will display. What it displays is the ones value of the given number.

Listing 3: Assembly Code for Part3

```
timer_INT   mov.w  #0000001000010000b, TA0CTL
            mov.w  #10486d, TA0CCR0
            mov.w  #0000000000010000b, TA0CCTL0
            clr    &TAIFG
            eint


TISR        dint
            clr    &TAIFG
            eint
            reti
```

PART3:

timer_INT: This section is set up for Timer Interrupt. Bits 9-8 indicate the signal selection, and in this experiment, we used the SMCLK signal. To achieve this, we set bits 9-8 to '10.' The frequency of the SMCLK signal is 1048576 Hz. We set TA0CCR0 to 10486 to generate interrupts with a 10-millisecond period. The 8th bit of TA0CCTL0 is set to 0 for compare mode, and the 4th bit is set to 1 to enable interrupts. TAIFG is set when an interrupt is pending. Since there is no interrupt, we cleared the interrupt flag. Finally, we enabled the interrupt with 'eint'.

TISR: This branch is called when an interrupt occurs. So, we start with disabled interrupts to prevent conflicts with another interrupt. We clear the interrupt flag and enable the interrupt as we finish processing the interrupt. Then, we return to the last instruction.

Listing 4: Assembly Code for Part4

```
timer_INT   mov.w  #0000001000010000b, TA0CTL
            mov.w  #10486d, TA0CCR0
            mov.w  #0000000000010000b, TA0CCTL0
            clr  &TAIFG
            eint


init_INT    bis.b  #007h, &P2IE
            and.b  #0BFh, &P2SEL
            and.b  #0BFh, &P2SEL2
            bis.b  #040h, &P2IES
            clr  &P2IFG
            eint



Setup_int   mov.b #0d, R10 ; csecond
            mov.b #0d, R11 ; second
            mov.b #11111111b, &P1DIR
            mov.b #00000000b, &P1OUT
            mov.b #00001111b, &P2DIR
            mov.b   #0d, R12
            mov.b   #0d, R13

Convertion  cmp       #00001010b, R10
            jl  Convertion2
            sub.b     #00001010b, R10
            inc.b R12
            jmp  Convertion
Convertion2 cmp       #00001010b, R11
            jl    Display1
            sub.b     #00001010b, R11
            inc.b R13
            jmp   Convertion2

Display1    clr   P1OUT
            clr   P2OUT
            mov.b #00000001b, R6
            mov.w #numbers, R7
            add   R12, R7
            mov.b 0(R7), P1OUT
            mov.b R6, P2OUT

Display2    clr   P1OUT
            clr   P2OUT
            rla   R6
```

```
            mov.w #numbers, R7
            add   R10, R7
            mov.b 0(R7), P1OUT
            mov.b R6, P2OUT

Display3    clr   P1OUT
            clr   P2OUT
            rla   R6
            mov.w #numbers, R7
            add   R13, R7
            mov.b 0(R7), P1OUT
            mov.b R6, P2OUT

Display4    clr   P1OUT
            clr   P2OUT
            rla   R6
            mov.w #numbers, R7
            add   R11, R7
            mov.b 0(R7), P1OUT
            mov.b R6, P2OUT
            jmp   Display1


ISR         dint
            cmp #00000001, &P2IN
            jeq start_T
            cmp #00000010, &P2IN
            jeq stop_T
            cmp #00000100, &P2IN
            jeq reset_T
start_T     mov  #9999d, &TA0CCR0
            jmp end_ISR
stop_T      mov  #0d, &TA0CCR0
            jmp end_ISR
reset_T     mov #0d,R10
            mov #0d,R11
end_ISR     clr &P2IFG
            eint
            reti


TISR        dint
            add #1d, R10
            cmp #99d, R10
            jl  TISRend
addSec      add #1d, R11
            mov #0d, R10
            cmp #99d, R11
            jl  TISRend
resetSec    mov #0d, R11
TISRend     clr   &TAIFG
            eint
            Reti


numbers   .byte 00000110b, 01011011b,
01001111b, 01100110b, 01101101b,
01111101b, 00000111b,  01111111b,
01101111b
endOfNumbers
```

PART4:

timer_INT: This section is set up for Timer Interrupt. Bits 9-8 indicate the signal selection, and in this experiment, we used the SMCLK signal. To achieve this, we set bits 9-8 to '10.' The frequency of the SMCLK signal is 1048576 Hz. We set TA0CCR0 to 10486 to generate interrupts with a 10-millisecond period. The 8th bit of TA0CCTL0 is set to 0 for compare mode, and the 4th bit is set to 1 to enable interrupts. TAIFG is set when an interrupt is pending. Since there is no interrupt, we cleared the interrupt flag. Finally, we enabled the interrupt with 'eint'.

init_INT: In this section, the interrupt initialization is configured. The interrupt flag is cleared, and the interrupt is enabled. .

Setup_int: The R10 register stores centiseconds, and R11 stores seconds. Initially, the register values are zero. The program initializes the I/O ports according to the port connections of the 7-segment display, which can be seen in the code. All bits of Port1 are set as output and will be used to control which character is currently being displayed. Bits 3-0 of Port2 are set as output and will be used to select which 7-segment display will currently display a character. To display two-digit numbers, R12 stores the tens place of centiseconds, and R13 stores the tens place of seconds.

Convertion & Convertion2: These functions find the digits of a 2 digit-number.

Displays: These functions clear the current display. R6 keeps the place of the cursor in the 7-segment display. Each time we rotate left to get the next index. R7 is set with the start address of numbers, and to find the corresponding number of digits, we add the value to it. Perform this operation for both the digits of seconds and centiseconds.

ISR: Three different buttons are used for start, stop, and reset operations. To stop, we set TA0CCR0 to 0, and to start, we set it to a random value. To reset, we clear registers R10 and R11, which store centiseconds and seconds.

TISR: Every 10 ms, an interrupt is generated. When the interrupt occurs 10 times, it equals 1 centisecond. When the centisecond reaches 100, the second is increased by 1. If the seconds reach 99, we reset the seconds since we cannot display 3-digit numbers.

III. RESULTS

For the first part, we successfully created an infinite loop as the main program code to light up different digits on the 7-segment display panel simultaneously. We achieved the same results to the sample output shown in Figure 1.

For the second part, we wrote a subroutine that converts binary input (ranging from 0x00000000b to 0x01100011b) to two decimal digits. We tried to the binary input is 97, and the subroutine correctly returns 9 and 7. We displayed these

Fig. 1: Result of Part-1

converted values on the 7-segment displays. For the third part, we wrote a time interrupt code. We configured the timer in compare mode with up-counting mode. For the fourth part, we tried to make a chronometer using the timer interrupt subroutine and BCD Conversion subroutine. We tried to save the centisecond and second values in two different variables. We assigned three buttons for reset, start, and stop. But the buttons did not work correctly.

IV. Discussion and Summary

In the first part of this project, we configured the seven-segment display to display 0123. In the second part, we wrote a subroutine that converts the given binary input (between 0x00000000b and 0x01100011b) to two decimals. We have tried for the binary input of 97 and the subroutine returned 9 and 7. We have displayed the returned value on the seven-segment display. In the third part, we created a timer interrupt. In the fourth part, we tried to set the timer interrupt to use the seven-segment display as a chronometer, showing centiseconds and seconds. We also tried assigning three different buttons reset, start, and stop. Our start and stop buttons were not working, but our reset button was working a little bit wrong. Our chronometer counted properly from 12 to 18, but then strange numbers appeared. No matter how hard we tried to run Part 3 properly, we could not manage to run it properly within the given time.