Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 6

Experiment Date : 20.11.2017

Group Number : Monday - 8

Group Members :

ID	Name	Surname
150150114	EGE	APAK
150140066	ECEM	GÜLDÖŞÜREN
150150701	YUNUS	GÜNGÖR

Laboratory Assistant : Ahmet Arış

1 INTRODUCTION

In this experiment, we expanded our knowledge about microprocessors and MSP430 Education Board with chronometer design. Using one of two timers on MSP430 micro-controller family, we developed a basic chronometer with centisecond accuracy. One centisecond is 0.01 seconds and 10 miliseconds. Using given documentation, we have determined control registers for timer unit and assigned them before using the timer. Also considering the frequency of given value we have calculated how many wave periods required for producing interrupts with period of 10 milliseconds.

Also, another thing to consider was producing output of 4 digits on 4 7-segment displays simultaneously. The trick is to displaying numbers one by one on each display very quickly. If the frequency is high enough output will seem like simultaneous.

In the first part of the experiment we have demonstrated simultaneous output for 4 different 7-segment displays.

In the second part, using code written in first part and timer, a chronometer was implemented. Counting numbers was displayed on 7-segment displays simultaneously.

2 EXPERIMENT

2.1 PART 1

In this part of the experiment, have demonstrated 4 digit output, using 4 different 7-segment displays simultaneously. First of all we have set output flags for P1 and P2 All of pins on P1 set as output and only least significant 4 bits of P2 set as output. Because P1 defines which part of a 7-segment display lights up, and least significant 4 bits of P2 determines which 7-segment display lights up according to P1.

To achieve simultaneity, we have implemented an infinite loop for displaying required digit on each one of screens in order. This loop tagged with “LOOP” in Picture 2.1.1. To display each digit correctly on 7-segment display we have used given array in data section. This array has data for representing number on 7-segment display in order. For example array’s first element can be used for displaying zero, second element can be used for displaying one and so on. Data in specific element of array was written to the P1 to light up correct parts of 7-segment display. R14 register used for holding data about current number. Moving array’s first element’s address to R14 and then incrementing R14 concurrently provided addresses that contains 7-segment display data for 0,1,2 and 3. Using array’s address directly on R14 let us create a more efficient solution. After writing data to P1 we have written necessary bits to P2 to light up one display each time.

After lighting up 4 7-segment displays simultaneously, we have realized that we did not turn of displays instantly, which caused a collision between different numbers and created an unreadable output. To fix that, we have assigned 0 bits for least significant four bits on P2 and turn of each display before writing data to P1. After writing data to P1, we have light up one of the displays with setting only one bit of P2. Then we have turned of every screen and repeated that process for each display in infinite loop.

```
                                        ; and retain current section.
            .retainrefs                 ; And retain any sections that have
                                        ; references to current section.

;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP     ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer


;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
            mov.b #0FFh, P1DIR
            mov.b #00Fh, P2DIR
LOOP        mov   #array, R14
            mov.b #000h, P2OUT
            mov.b @R14, P1OUT
            mov.b #001h, P2OUT
            inc R14
            mov.b #000h, P2OUT
            mov.b @R14, P1OUT
            mov.b #002h, P2OUT
            inc R14
            mov.b #000h, P2OUT
            mov.b @R14, P1OUT
            mov.b #004h, P2OUT
            inc R14
            mov.b #000h, P2OUT
            mov.b @R14, P1OUT
            mov.b #008h, P2OUT
            jmp LOOP


array   .byte   00111111b, 00000110b, 01011011b, 01001111b, 01100110b, 01101101b, 01111101b, 00000111b, 01111111b, 01101111b
lastElement

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
```

Picture 2.1.1 – code for experiment part 1

2.2 PART 2

In the second part of the experiment we have implemented a timer and a button to pause the chronometer. First of all we have set up memory locations for seconds and centiseconds values. While setting values on memory, we have written data block at the end of file which caused unchangeable values, and application did not run as expected. After consulting to our TA, we have implanted data block on the top which solved the issue.

Then we have set our output bits, and interrupt bits. P1 and P2 output bits explained in part 1 of this experiment. P1 and least significant 4 bits of P2 used for displaying output on 4 different 7-segment displays simultaneously. Also 6th bit of P2 set for interrupted signal. Pressing that button will cause an interrupt in microprocessor to pause chronometer. Process on P2IE, P2SEL, P2SEL2, P2IES and P2IFG provides interrupt functionality on 6th bit of P2. Also interrupted vector of P2.6 defined as ISR.

After setting output and interrupt flags, we have set timer control flags. In this experiment timer A in MSP430 microprocessor was used. We have configured our timer for producing an interrupt when counter reached 10486. This value calculated considering given waves frequency. Given wave is a square wave with frequency of 1048576Hz. It is named as SMCLK in MSP430 microprocessor documents. We have set 9th and 4th bits and cleared rest of bits of timer's control register named as TA0CTL. This bit set configures timer to use SMCLK wave which is explained below, and configures to run in up mode. In up mode counter counts up to TA0CCR0 on each pulse of the configured wave. To create an output on each 10 milliseconds TA0CCR0 was set as 10486. This value calculated according to frequency of wave. Then we have set 4th bit of TA0CCTL0 to enable interrupts. Interrupt vector for timer set as TISR.

On each interrupt from timer we have incremented centiseconds value on memory. If centisecond reached value of 100, centiseconds set to 0 and second incremented by one.

To display values of seconds and centiseconds values on 7-segment displays, we have used code from part 1 of this experiment. This part named as DISPLAY in the code. There was only two modifications to code from part 1. Calling BCD subroutine for dividing seconds and centiseconds values to digits. This subroutine called before setting first and second displays for seconds, and before setting third and fourth displays for centiseconds.

BCD subroutine takes input from R15 and returns output on R14 and R15. R14 represent higher value digit and R15 represent lower value digits. Keep in mind data digits on R14 and R15 are decimal together. When used in 7-segment displays it creates a human readable output. BCD subroutine works by subtracting 10 in decimal or A in hexadecimal concurrently if number is bigger then 10 in decimal. At each subtraction from R15, R14 incremented by one.

At each interrupt from P2.6 timer interrupt enable flag which is 4^{th} bit of TA0CCTL0 toggled. This means that if interrupt enabled it is disabled and counting process stops. If interrupt was disabled it is enabled again and counting process continues.

More information can be found on code

```
1  ;-------------------------------------------------------------------------------
2  ; MSP430 Assembler Code Template for use with TI Code Composer Studio
3  ;
4  ;
5  ;-------------------------------------------------------------------------------
6              .cdecls C,LIST,"msp430.h"       ; Include device header file
7
8  ;-------------------------------------------------------------------------------
9              .def    RESET                   ; Export program entry-point to
10                                             ; make it known to linker.
11 ;-------------------------------------------------------------------------------
12
13             .data
14 seconds          .byte   00h
15 centiseconds     .byte   00h
16
17             .text                           ; Assemble into program memory.
18             .retain                         ; Override ELF conditional linking
19                                             ; and retain current section.
20             .retainrefs                     ; And retain any sections that have
21                                             ; references to current section.
22
23 ;-------------------------------------------------------------------------------
24 RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
25 StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
26
27
28 ;-------------------------------------------------------------------------------
29 ; Main loop here
30 ;-------------------------------------------------------------------------------
31             mov.b #0FFh, P1DIR
32             mov.b #00Fh, P2DIR
33             bis.b #040h, &P2IE
34             and.b #0BFh, &P2SEL
35             and.b #0BFh, &P2SEL2
36
37             bis.b #040h, &P2IES
38             clr &P2IFG
39
40             mov     #0210h, TA0CTL
41             mov     #28F6h, TA0CCR0
42             mov     #010h,  TA0CCTL0
43             eint
44
45 ST          call #DISP
46             jmp ST
47
48 ISR         dint
49             clr &P2IFG
50             xor     #010h,  TA0CCTL0
51             eint
52             reti
53
54
55
56 TISR        dint
57             inc.b &centiseconds
58             cmp.b #064h, &centiseconds
59             jl RETTISR
```

Picture 2.2.1 – code between lines 1-59 for experiment part 2

```
54
55
56 TISR        dint
57            inc.b &centiseconds
58            cmp.b #064h, &centiseconds
59            jl RETTISR
60            inc.b &seconds
61            mov.b #00h, &centiseconds
62 RETTISR     eint
63            reti
64
65 DISP        mov.b &seconds, R15
66            call #BCD
67            mov #00h, P2OUT
68            mov.b array(R14), P1OUT
69            mov #01h, P2OUT
70            mov #00h, P2OUT
71            mov.b array(R15), P1OUT
72            mov #02h, P2OUT
73            mov.b &centiseconds, R15
74            call #BCD
75            mov #00h, P2OUT
76            mov.b array(R14), P1OUT
77            mov #04h, P2OUT
78            mov #00h, P2OUT
79            mov.b array(R15), P1OUT
80            mov #08h, P2OUT
81            ret
82
83
84 BCD         mov #00H, R14
85 LOOP        cmp #0Ah, R15
86            jl  CONT
87            sub #0Ah, R15
88            inc R14
89            jmp LOOP
90 CONT        ret
91
92
93 array   .byte   00111111b, 00000110b, 01011011b, 01001111b, 01100110b, 01101101b, 01111101b, 00000111b, 01111111b, 01101111b
94 lastElement
95
96 ;-------------------------------------------------------------------------------
97 ; Stack Pointer definition
98 ;-------------------------------------------------------------------------------
99            .global __STACK_END
100           .sect   .stack
101
102 ;-------------------------------------------------------------------------------
103 ; Interrupt Vectors
104 ;-------------------------------------------------------------------------------
105           .sect   ".reset"                ; MSP430 RESET Vector
106           .short  RESET
107           .sect   ".int09"
108           .short  TISR
109           .sect   ".int03"
110           .short  ISR
111
112
```

Picture 2.2.2 – code between lines 54-112 for experiment part 2

3 CONCLUSION

In this experiment we have learned that using different output devices simultaneously by using them one by one very fast. Also, we have learned more about control ports, examining documentation of different devices, output ports and interrupts. We have seen that multiple interrupts are possible, and we have learned how to implement that. We have also remembered that how to implement memory variables.