Experiment 5

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

***Abstract*—This report details the implementation of an experiment focusing on 7-segment displays and interrupt subroutines using the MSP430 microcontroller. The experiment involves sequentially displaying digits, letters, and a combination of both on a 7-segment display. Additionally, an interrupt-driven mechanism is created to dynamically switch between different display modes and reset or pause a digit counter.**

***Index Terms*—7-segment display, LEDs, ports, digits, letters, counter, GPIO, interrupt, interrupt subroutine, push button**

I. Introduction

In this experiment two new concepts are introduced: 7-segment display and interrupt subroutine. The main goal is to create a program that counts between 0-9, displays letters sequentially, dynamically transitions between various display modes using interrupts, and resets/pauses/unpauses counting using again interrupts. The hardware setup involves configuring I/O ports to control the 7-segment display and handle the button presses. In the end, a comprehensive exploration of 7-segment display control and interrupt handling using the MSP430 microcontroller is expected to be achieved.

II. Materials and Methods

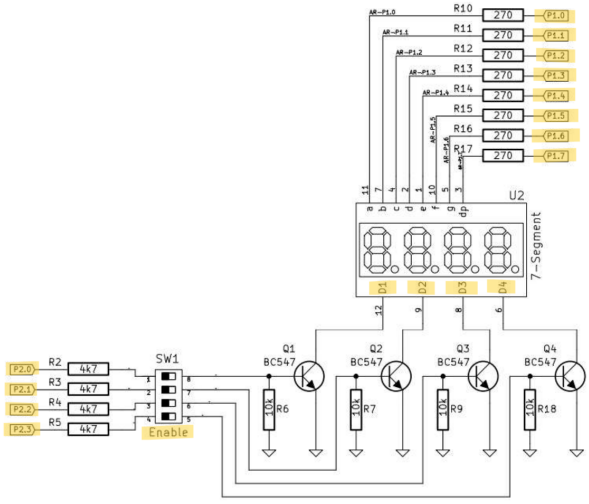

Fig. 1: Port Connections of 7-Segment Display

PART1:

```
Setup       mov.b   #11111111b, &P1DIR
            mov.b   #00000000b, &P1OUT
            mov.b   #00001111b, &P2DIR
            mov.b   #00000001b, &P2OUT

Reset       mov.w   #numbers, R4

Display     mov.b   0(R4), P1OUT
            call    #Delay
            add     #1d, R4
            cmp     #endOfNumbers, R4
            jeq     Reset
            jmp     Display

Delay       mov.w   #0Ah, R14
L2          mov.w   #07A00h, R15
L1          dec.w   R15
            jnz     L1
            dec.w   R14
            jnz     L2
            ret

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

Setup: It initializes the I/O ports according to port connections of 7 segment display which can be seen below. 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.

.data: In the data section there is a label called 'numbers'. It represents the start of an array that holds binary values for bit pattern of digits from 0 to 9. When this pattern is given to P1OUT, corresponding digit is shown on the 7-segment display. 'endOfNumbers' label is for indicating the end of the array.

Reset: The program counts between 0-9 'repeatedly'. So each time, including the first time, the start address of the array should be stored in a register, R4 in this case.

Display: In this subroutine for each new digit the memory location pointer R4 is increased by 1. In this way an offset specific to the digit intended to display is added to the start address of the array 'numbers'. The bit pattern for the digit is obtained and then send to PORT1 as output. In order to display current digit for a while, Delay subroutine is called. R4 is checked each time. If it is reached to the end address of the array, jump to Reset occurs to turn back to the start address.

Delay: This is a Delay subroutine which is called each time a new character is set to be displayed. It takes something like 1 second, so counting between 0-9 lasts broadly 10 seconds.

PART2:

```
Setup       ...

Reset       mov.w   #letters , R4

Display     ...
            cmp     #endOfLetters , R4
            ...

Delay   ...

            .data
letters   .byte  01110111b, 00111001b,
01110110b, 00110000b, 00111000b, 00111000b,
01111001b, 00111110b, 01101101b
endOfLetters
```

The only difference between this part of the experiment and the first part of the experiment is letters being used instead of digits. So the code from Part1 is arranged accordingly. This time, the data section is filled with bit patterns of letters A-C-H-I-L-L-E-U-S.

PART3:

```
Setup       mov.b   #11111111b, &P1DIR
            mov.b   #00000000b, &P1OUT
            mov.b   #00001111b, &P2DIR
            mov.b   #00000001b, &P2OUT
            mov.b   #mode, R6
            mov.b   #001h, (R6)
            mov.b   #0d, R7

init_INT    bis.b   #040h, &P2IE
            and.b   #0BFh, &P2SEL
            and.b   #0BFh, &P2SEL2

            bis.b   #040h, &P2IES
            clr     &P2IF
            eint ; enable interrupts

Reset       mov.w   #letters , R4
            mov.w   #numbers , R5
```

I/O Ports are initialized as in previous parts.

Address of the mode variable is put into R6.

Mode variable is initialized with 1 by reaching it with pointer R6.

R7 is set as 0 initially. This will be used as a flag for mode transitions.

Interrupt initialization is done with standard configurations. Interrupt is enabled at 6th bit of Port2.

The start addresses of 'letters' and 'numbers' arrays are put into R4 and R5 respectively.

```
Control     cmp     #004h, (R6)
            jeq     resetMode
            cmp     #1d, R7
            jeq     ResetForChange
            cmp     #001h, (R6)
            jeq     Display1
            cmp     #002h, (R6)
            jeq     Display2
            cmp     #003h, (R6)
            jeq     Display3
            jmp     Control
```

There is a control mechanism here. If mode variable pointed by R6 tries to reach 4, jumping to resetMode subroutine does not allow and returns it back to 1. So there is only three modes: 1, 2 and 3.

```
resetMode   mov.b   #001h, (R6)
            jmp     Control
```

The flag for mode transition is checked. If it is 1, then jumped to ResetForChange subroutine, which turns memory addresses hold by R4 and R5 back to the start addresses of the arrays. The flag inside R7 is also reset.

```
ResetForChange
            mov.w   #letters , R4
            mov.w   #numbers , R5
            mov.b   #0d, R7
            jmp     Control
```

Then the display mode is checked. If it is 1, jumped to Display1. If it is 2, jumped to Display2. If it is 3, jumped to Display3. All Display subroutines jumps to Control back after 1 second of Display delay.

In Display2 mode, bit pattern for current digit value is selected from the array by using address R4. Then this address is increased by one. If it is reached to the end address of the array, jumped to the Reset to turn memory addresses hold by R4 and R5 back to the start addresses of the arrays. Otherwise it jumps to Control. Each time this subroutine is executed, only one digit is displayed. As long as an interrupt does not occur, digits between 1-9 are displayed one after another by stopping by Display1 each time. After 9, program returns to 1 and counts between 1-9 again.

In Display1 mode, everything happens exactly the same. But the only difference here is displayed characters are A-C-H-I-L-L-E-U-S respectively.

In Display3 mode, the display sequence is 1-A-2-C-3-H-4-I-5-L-6-L-7-E-8-U-9-S the combination of the other two modes'. Here, the 'numbers' and 'letters' arrays are used together. First time, the first digit 1 and the first letter A is taken and displayed one after another. The next time, the next digit and the next letter is taken and displayed one after another. Other than that, this also works exactly the same.

```
Display1    mov.b   0(R4), P1OUT
            call    #Delay
            add     #1d, R4
            cmp     #endOfLetters, R4
            jeq     Reset
            jmp     Control

Display2    mov.b   0(R5), P1OUT
            call    #Delay
            add     #1d, R5
            cmp     #endOfNumbers, R5
            jeq     Reset
            jmp     Control

Display3    mov.b   0(R5), P1OUT
            call    #Delay
            add     #1d, R5
            mov.b   0(R4), P1OUT
            call    #Delay
            add     #1d, R4
            cmp     #endOfLetters, R4
            jeq     Reset
            jmp     Control
```

In Interrupt Subroutine, mode counter is increased by one. And also mode transition flag is set to 1 (by increasing 0 by 1, R7 is reset to 0 every time before interrupt).

```
ISR         dint
            add     #1d, (R6)
            inc.b   R7
            clr     &P2IFG
            eint
            reti
```

Here the mode variable and the arrays that hold bit patterns for digits and letters were declared in the data area.

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

letters    .byte 01110111b, 00111001b,
01110110b, 00110000b, 00111000b, 00111000b,
01111001b, 00111110b, 01101101b
endOfLetters

mode       .space  1
```

PART4:

```
Setup       ... ; Initialize I/O Ports

init_INT    ... ; Initialize interrupt
                  configuration

ResetCounter
            mov.b   #0d, R5 ;push counter
            mov.w   #numbers, R4
            jmp     Display

Reset       mov.w   #numbers, R4

Display     mov.b   0(R4), P1OUT
            call    #Delay
            add     #1d, R4
            cmp     #5d, R5
            jeq     ResetCounter
            cmp     #endOfNumbers, R4
            jeq     Reset
            bit.b   #00000001, R5
            jne     Pause
            jmp     Display

Delay       mov.w   #0Ah, R14
L2          mov.w   #07A00h, R15
L1          dec.w   R15
            jnz     L1
            dec.w   R14
            jnz     L2
            ret

Pause       bit.b   #00000001, R5
            jeq     Display
            jmp     Pause

ISR         dint
            add     #1d, R5
            clr     &P2IFG
            eint
            reti

            .data

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

ResetCounter: Push counter is reset. The start address of the 'number' array is put into R4. Jumped to display. These are done once at the beginning and each time the selected button is pressed for the 5th time.

ISR: In interrupt subroutine, the push counter is increased by 1 each time button is pressed.

Delay: The current digit whose 7-segment display bit pattern is selected from array by the current value of R4. R4 is increased for the next digit to be displayed. If R4 is reached to the end of the array, then it is set to the start address again in Reset subroutine. But before that there is one more privileged thing to check: If push button is pressed 5th time jumped to the ResetCounter in order to reset the process and start from the first number. If the button is pressed 1st or 3rd times, which are odd, jumped to Pause and the counter is paused.

Pause: It is checked if the button is pressed 2nd or 4th times, which are even, then jumped to the Display and the counter is unpaused. Otherwise, subroutine becomes a loop by constantly jumping to itself. This ensures the pause state.

III. Results

Part 1 - Counting Digits:

A counter program that displayed digits 0-9 in a loop is implemented. A delay subroutine is utilised in order to ensure that each digit was shown for approximately one second, resulting in a complete counting cycle which lasts around 10 seconds. Results are observed from the 7-segment display of the MSP430 Microcontroller board.

Part 2 - Displaying Letters:

This part is achieved by building on the first part. The code is modified to display the letters ”A-C-H-I-L-L-E-U-S” sequentially instead of digits 0-9. Results are observed from the 7-segment display of the MSP430 Microcontroller board.

Part 3 - Interrupt-Driven Mode Switching:

In the third part interrupt-driven mode switching is introduced. It is allowed to dynamically transit between three display modes: displaying letters only, numbers only, and a combination of both. Mode switches are triggered by button presses and this is handled by interrupt subroutine. This showed the flexibility of the MSP430 microcontroller in adapting to different display requirements. Interrupts, so the mode switches are triggered by pushing the 6th bit of Port2 of the board and the results are observed from 7-segment display.

Part 4 - Interrupt-Driven Counter Reset/Stop:

In the final part, a counter reset and stop mechanism was implemented using interrupts. Button presses are handled by interrupt subroutine so the the counting process is reset after the 5th button press, is paused after the 1st and 3rd button presses and is unpaused after the 2nd and 5th button presses. Interrupts, so the reset, pause and unpause operations are triggered by pushing the 6th bit of Port2 of the board and the results are observed from 7-segment display.

IV. Discussion and Summary

During this experiment we gained a broad insight into capabilities of 7-segment displays and interrupt handling on the MSP430 microcontroller. How interrupt works and how the rest of the code flows when interrupt subroutine is not executed is a bit confusing to us, at first. But, with continuously improving the code and debugging it we got the idea. We successfully implemented, executed and debugged the codes for each specific tasks and reached the goal of the experiment.