ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO : 6

EXPERIMENT DATE : 20.12.2024

LAB SESSION : FRIDAY - 14.30

GROUP NO : G9

GROUP MEMBERS:

150220723 : Abdulsamet Ekinci

150210906 : Emil Huseynov

150220902 : Nahid Aliyev

FALL 2024

Contents

1 INTRODUCTION [10 points]

In this experiment, we explored the timer functionality of the MSP430 microcontroller to implement a chronometer and a 7-segment display driver. The experiment was divided into two parts: the first involved displaying digits simultaneously using high-frequency refresh, while the second focused on developing a fully functional chronometer with interrupt-driven input handling. This provided an opportunity to understand timer configuration and interrupt-based programming.

2 MATERIALS AND METHODS [40 points]

This section describes the hardware, software, and methods used in the experiment.

2.1 Hardware

- **MSP430G2553 Microcontroller:** A low-power microcontroller with a 16-bit architecture and support for interrupts.
- **7-Segment Display:** Used for numerical output display.
- **Buttons:** Three buttons configured for reset, start/stop, and save-best-time functionalities.

2.2 Software

- **Code Composer Studio (CCS):** IDE used to write, debug, and flash assembly code to the MSP430.

2.3 Part 1: 7-Segment Display Refresh

Objective: To display four digits on a 7-segment display using high-frequency switching to achieve simultaneous output.

Procedure:

- **Step 1: Initialization** - Configure GPIO ports for 7-segment display and digit selection.
- **Step 2: Main Loop** - Implement an infinite loop cycling through the digits at high frequency.
- **Step 3: Short Delay** - Introduce a delay subroutine to avoid ghosting effects.

Code Snippet:

```
mov.b   array(R5), &P1OUT ; Output digit to 7-segment
mov.b   #0x01, &P2OUT     ; Activate corresponding digit
call    #ShortDelay       ; Short delay for visibility
```

Results: The display correctly showed all four digits simultaneously due to high-frequency cycling.

2.4 Part 2: Chronometer Implementation

Objective: Develop a chronometer with start, stop, reset, and save-best-time features using interrupts.

Procedure:

- **Step 1: Timer Configuration** - Configure the timer in up mode with SMCLK as the clock source.
- **Step 2: Interrupt Handling** - Write ISRs for both timer and button presses to handle the chronometer functionalities.
- **Step 3: BCD Conversion** - Implement a subroutine to convert time values into BCD for display on the 7-segment.

Challenges and Solutions:

- **Interrupt Handling:** Initial implementation caused recursive ISR calls due to improper flag handling. This was resolved by clearing interrupt flags correctly.
- **BCD Conversion:** Proper conversion of time values to BCD format ensured correct output display.

Code Snippet:

```
TIMER_ISR:
    inc.b   centiseconds
    cmp.b   #100, centiseconds
    jne     skip_seconds
    clr.b   centiseconds
    inc.b   seconds
skip_seconds:
    call    #BCD_Conversion
    bic.w   #CCIFG, &TA0CCTL0 ; Clear interrupt flag
    reti
```

3 RESULTS AND DISCUSSION [30 points]

In the first part of the experiment, the implementation was successful. The 7-segment display accurately showed digits 0-3 simultaneously, demonstrating the effectiveness of high-frequency digit refresh. The approach of cycling through the digits at a rapid pace allowed the display to appear seamless to the human eye, meeting the objectives of this portion of the experiment.

However, the second part of the experiment was not successful. Despite significant effort, the chronometer functionality could not be achieved due to several persistent issues. One major challenge was improper interrupt handling, which caused recursive calls to the interrupt service routine (ISR) and resulted in stack overflow. Additionally, errors in timer configuration, such as incorrect register settings, prevented the timer from generating interrupts at the required 10ms intervals. These problems impeded the operation of the chronometer.

Even after debugging and attempting to address these issues by revising the interrupt flag handling and correcting the timer and GPIO configurations, the chronometer still did not function as intended. Specifically, the start, stop, reset, and save-best-time features did not respond reliably to button inputs. These challenges highlighted the complexity of working with timers and interrupts in assembly language, underscoring the need for further refinement of the implementation and debugging strategies. Despite the setbacks, this part of the experiment provided valuable insights into the intricacies of timer-based operations and interrupt-driven programming on the MSP430 microcontroller.

4 CONCLUSION [20 points]

This experiment emphasized the importance of timer and interrupt configurations in microcontroller programming. While the first part demonstrated seamless high-frequency digit refresh, the second part highlighted common pitfalls in interrupt-driven designs, such as flag handling and stack management. Resolving these issues deepened our understanding of assembly programming on the MSP430.