Lab #1
Foundations of Embedded Systems

ECE2049 - Embedded Computing In Engineering Design
Filippo Marcantoni & Anthony Previte
01/27/2023

Introduction:

In this lab we created our first game using the MSP430F5529 Launchpad-based Lab Board. We were able to make a Simon program where LEDS would light up and a user would input corresponding numbers on the keypad to match the sequence. We also configured the buttons on the board to act as a reset button at any moment in the program. We implemented this using C and practiced using state machines, port configurations, and writing functions.

Discussion and Results:

Pre Lab) For the pre lab we looked over the assignment and created a rough draft of how our code would go. In this document we had main, a forever while loop, a state machine, and pseudocode for what the game was supposed to do in the correct spots such as lighting the LEDS, getting the input from the user, checking if the sequence was right, and so on.

1) First we created the welcome screen in state 0, which displays SIMON on the LCD using the draw string function.

2) Still in state 0, we made an if statement where if the current key being pressed on the keypad is '*', then a countdown from 3 to 1 begins and then the game starts as the state changes to 1.

3) In state 1, we generate one random number from 1 to 4 and we save it in the array num[32] so that we can keep track of the sequence. Each number corresponds to a respective LED, the left-most LED is "1" and the right-most is "4".
   Then the sequence that was generated is displayed on the LEDs and the buzzer sounds with each LED flash and the buzzer has different pitches for each LED. After the LEDs are finished playing the state switches to 2.

4) In state 2, the user inputs the sequence of LEDs that flashed using the keypad and the program saves it in the array input[32]. The user can press any key here, but keys outside the range of 1-4 result in a printed error message. When the user presses a key 1-4 the number will show up with a spatial alignment on the display for a short period of time, where 1 is left most and 4 is right most. Once the number of inputs matches the number of LED's that lit up the state changes to 3.

5) After each successful round the sequence gets longer and faster. The sequence's length and speed is incremented at the end of state 3, which compares the values of each position of the arrays num, containing the LEDs' sequence, and input,

containing the sequence typed in by the user. If the two arrays contain the same values so that the player typed the correct sequence, the variable x, which declares the length of both sequences for the level that the player is on, is incremented by 1. We did this using the variable in all of ours for loops: $for\,(i\ =\ 0,\ i\ <\ x,\ i\ ++)$. Since our two arrays have a length of 32, we made an if statement once the x is incremented that checks the value of x: if x is bigger than 32, it means that no values can be stored anymore in our two arrays, which means that the player has beaten the game and perhaps has won.
To increase the speed of the sequence we used a variable t as the input of the software delay function swDelay, which is applied every time a LED is turned on, and, if the sequence is correct, the variable t decreases by 0.025.

6) We used the rand() function and put it in our state 1 where the LEDs are lit. There is one new number generated each time this state is reached and it is added to the end of our sequence array so that the previous sequence is the same only with one additional number at the end of it now.

7) We set our maximum sequence length to 32. We needed this maximum length to initialize our two arrays num[32] and input[32] and to save the sequence displayed by the LEDs and the sequence typed in by the user. We also need it because in the moment that a user reaches level 32, no more values can be stored in our two arrays which means that the player has won and after a congratulation message on the LCD state switches back to 0 and the game can be replayed.

8) We modified our Buzzer on function to take an input, period length, to change the pitch of the buzzer. In contrast to the original buzzer on function where it set `TB0CCR0 = 128;` we made it `TB0CCR0 = (32768/period);` which changed the PWM in ACLK ticks. This allows the frequency of the buzzer to change based on the input period. We then called this function with a different input for each buzzer when its corresponding LED lit up in state 1.

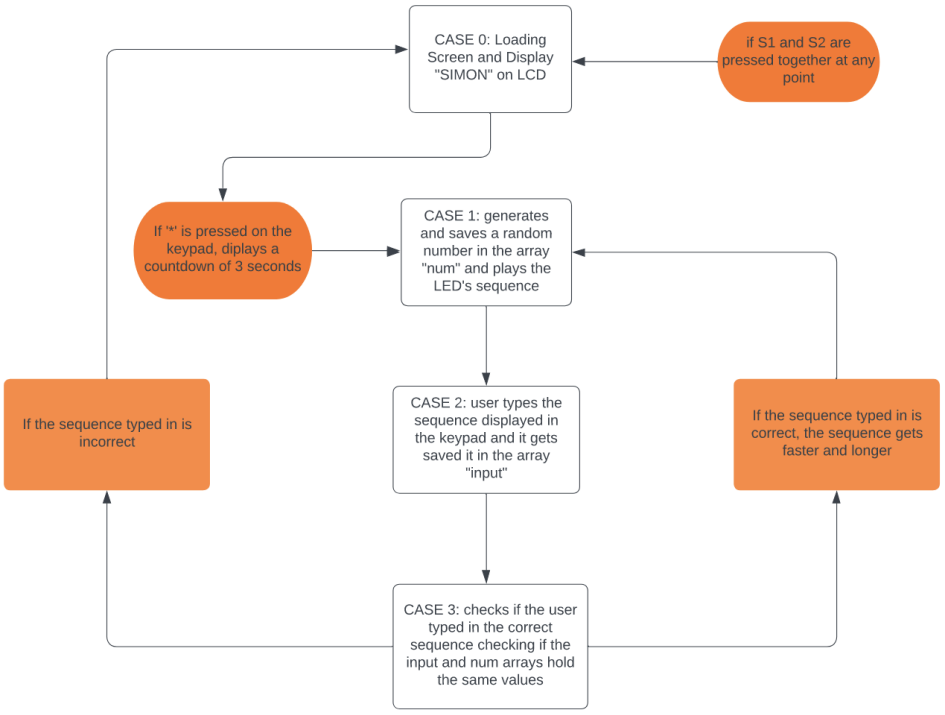
9) We wrote functions to configure the buttons S1 and S2 and detect when they are being pushed in addition to our new buzzer on function. The function setup() configures the 2 buttons, by making them digital, then inputs, a pullup resistor to counter the ground the button has when pressed, and finally allowing them to have outputs as well. The function pressed() checks if the buttons are pressed and returns an 8 bit mask called “on” that corresponds to the buttons being pressed.
Our functions are below.

```c
void Buzzer2(unsigned char per)
{
    // Initialize PWM output on P3.5, which corresponds to TB0.5
    P3SEL |= BIT5; // Select peripheral output mode for P3.5
    P3DIR |= BIT5;

    TB0CTL  = (TBSSEL__ACLK|ID__1|MC__UP);  // Configure Timer B0 to use ACLK, divide by 1, up mode
    TB0CTL  &= ~TBIE;                       // Explicitly Disable timer interrupts for safety

    // Now configure the timer period, which controls the PWM period
    // Doing this with a hard coded values is NOT the best method
    // We do it here only as an example. You will fix this in Lab 2.
    TB0CCR0   = (32768/per);                    // Set the PWM period in ACLK ticks
    TB0CCTL0 &= ~CCIE;                  // Disable timer interrupts

    // Configure CC register 5, which is connected to our PWM pin TB0.5
    TB0CCTL5  = OUTMOD_7;                   // Set/reset mode for PWM
    TB0CCTL5 &= ~CCIE;                      // Disable capture/compare interrupts
    TB0CCR5   = TB0CCR0/2;                  // Configure a 50% duty cycle
}
```

```c
void setup(){
    P7SEL &= ~ (BIT0);    //bit 4 and 0 are 0
    P3SEL &= ~ (BIT6);    //bit 6 is 0
    //P2SEL &= ~ (BIT2);    //bit 2 is 0

    P7DIR &= ~ (BIT0);
    P3DIR &= ~ (BIT6);
    //P2DIR &= ~ (BIT2);

    P7REN |= (BIT0);
    P3REN |= (BIT6);
    //P2REN |= (BIT2);

    P7OUT |= (BIT0);
    P3OUT |= (BIT6);
    //P2OUT |= (BIT2);

}
```

```c
int pressed(){
    int on = 0x00;
    int S1 = P7IN&BIT0;
    int S2 = P3IN & BIT6;

    if (S1 ==0){
        on = (on|BIT0);
    }
    if (S2 ==0){
            on = (on|BIT1);
       }
    return on;
}
```

10) Flowchart of our code)



Conclusion:

We were successful in completing the lab objectives. We had a functional Simon game and completed the bonuses by changing the pitch of each LED's buzzer and making the time between LEDS faster as you navigate through successful rounds. We now have our first complete project done and have a sense of how a simple game should operate using the lab board. We also gained experience using state machines in C and configuring and using ports on the lab board.

Appendices:

ASCII Table used: https://www.ascii-code.com/