ECE 2049– Laboratory Report
Embedded Computing in Engineering Design
Worcester Polytechnic Institute
C-Term 2021
Section: C03
Laboratory 1

Submitted by

Jonathan R. Lopez

2/14/2021

Professor: Yarkin Doroz

Co-Instructor: Fatemeh (Saba) Ganji

Introduction:

The purpose of lab 1 is to create the classic game of Simon by Milton Bradley using our MSP430F5529, our keypad, our buzzer and our SHRP128 LCD screen. The goal of this lab is to play Simon using the numbers 1-4 on our keypad and positioning the numbers on our screen in order of what buttons need to be pressed. The player then must press the correct keys in order to proceed to the next level. Where one more number appears and the user must get the numbers in the correct sequence again to proceed. We will go up to 10 levels and use numbers 1-4. If there is any error the game end. In this lab we will be using the buzzer, the LEDs, the keypad and LCD screen to play this game.

Materials

TI MSP430F5529 Launchpad based lab board.
Jumper wires
Buzzer
Breadboard
Sharp128 LCD Screen
0-9, * and # Keypad
Micro-USB to USB-A cable
Computer with Code Composer Studio ver. 10.2

Discussion and Results:

Getting started:

With the MSP430F5529 already wired up from Lab 0, we got right into launching Code Composer Studio (CCS).

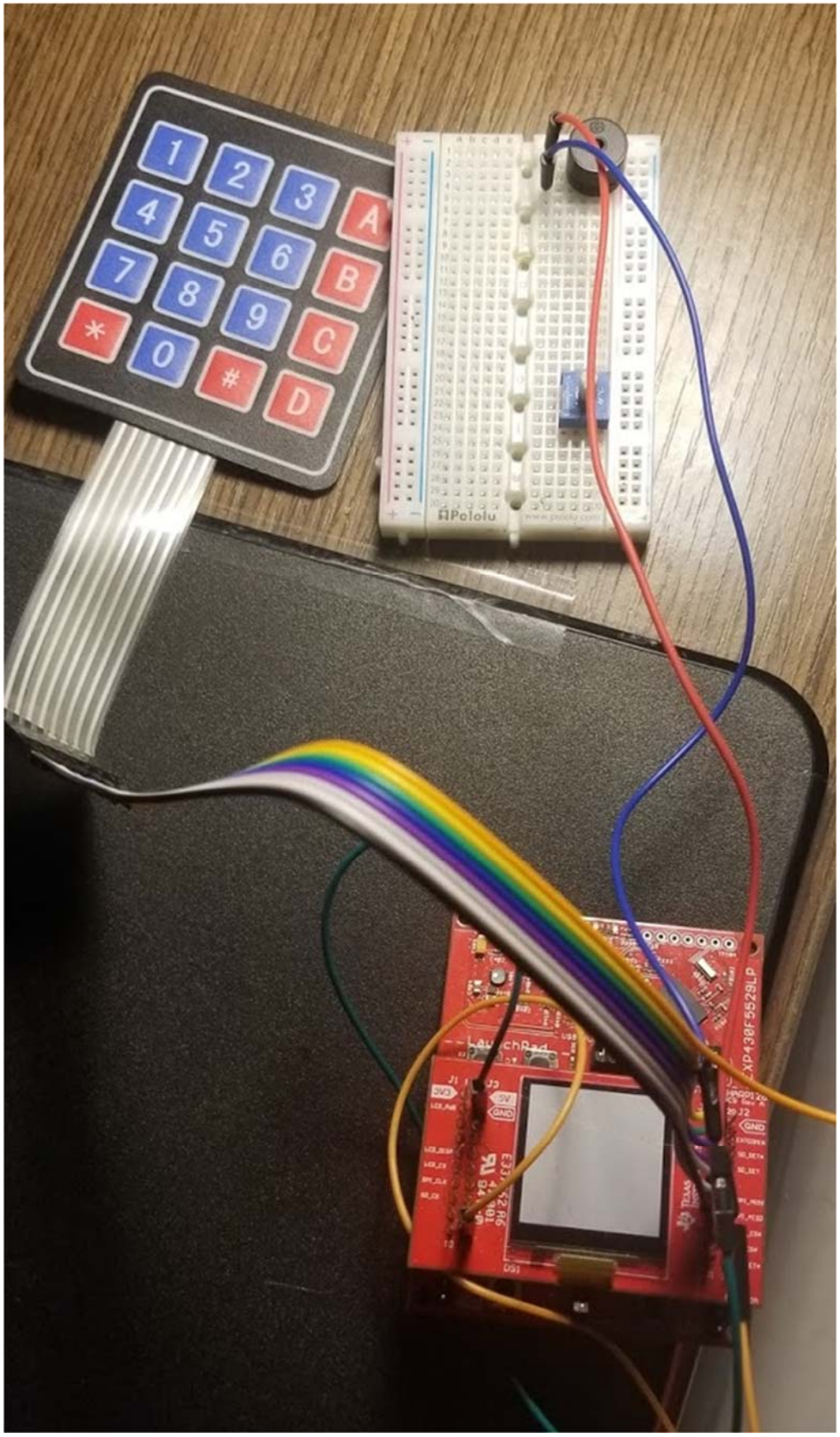

Figure 1: MSP430F5529 already wired up and ready for Lab 1

Approach:

As stated in the introduction, our goal is to make the game Simon on our MSP430F5529 using the LED lights, the buzzer and most importantly the keypad.

We will used numbers 1-4 to represent the four corners of the Simon game.”1” will displayed towards the left while 4 is all the way on the right. To check these values, we will use an array of integers with the size of 10 as said in the lab write up.

Why do you need to select a maximum length for the sequence?

You need select a maximum length for the sequence because it would keep increasing and use up all the memory addresses. 10 numbers are a good sequence to start with.

From the demo code we have the buzzer enable and LED setting to high and low functions already.

The buzzer buzzes due to a PWM (Pulse Width Modulated) signal is applied to that pin. Instead of getting rid of the BuzzerOn() function I created a new function called BuzzerPWM() in peripherals.c

```
72 void BuzzerOn(void)
73 {
74     // Initialize PWM output on P3.5, which corresponds to TB0.5
75     P3SEL |= BIT5; // Select peripheral output mode for P3.5
76     P3DIR |= BIT5;
77
78     TB0CTL  = (TBSSEL__ACLK|ID__1|MC__UP);  // Configure Timer B0 to use ACLK, divide by 1, up mode
79     TB0CTL  &= ~TBIE;                       // Explicitly Disable timer interrupts for safety
80
81     // Now configure the timer period, which controls the PWM period
82     // Doing this with a hard coded values is NOT the best method
83     // We do it here only as an example. You will fix this in Lab 2.
84     TB0CCR0   = 128;                    // Set the PWM period in ACLK ticks
85     TB0CCTL0 &= ~CCIE;                  // Disable timer interrupts
86
87     // Configure CC register 5, which is connected to our PWM pin TB0.5
88     TB0CCTL5  = OUTMOD_7;                   // Set/reset mode for PWM
89     TB0CCTL5 &= ~CCIE;                      // Disable capture/compare interrupts
90     TB0CCR5   = TB0CCR0/2;                  // Configure a 50% duty cycle
91 }
92
93 void BuzzerPWM(int pitch)
94 {
95
96     // Initialize PWM output on P3.5, which corresponds to TB0.5
97         P3SEL |= BIT5; // Select peripheral output mode for P3.5
98         P3DIR |= BIT5;
99
100        TB0CTL  = (TBSSEL__ACLK|ID__1|MC__UP);  // Configure Timer B0 to use ACLK, divide by 1, up mode
101        TB0CTL  &= ~TBIE;                       // Explicitly Disable timer interrupts for safety
102
103        // Now configure the timer period, which controls the PWM period
104        // Doing this with a hard coded values is NOT the best method
105        // We do it here only as an example. You will fix this in Lab 2.
106        TB0CCR0   = pitch;                    // Set the PWM period in ACLK ticks
107        TB0CCTL0 &= ~CCIE;                  // Disable timer interrupts
108
109        // Configure CC register 5, which is connected to our PWM pin TB0.5
110        TB0CCTL5  = OUTMOD_7;                   // Set/reset mode for PWM
111        TB0CCTL5 &= ~CCIE;                      // Disable capture/compare interrupts
112        TB0CCR5   = TB0CCR0/2;                  // Configure a 50% duty cycle
113 }
```

Figure 2: BuzzerPWM and BuzzerOn functions

Altering the function to accept wasn't particularly difficult. It now needs an integer input for the pitch you want. In the function body, I replaced the 128 with the integer parameter. In this case it's named pitch. This is for the PWM period in ACLK ticks. This is how you vary the pitch by adjusting the clock signal period.

Flow chart:
The best way to accomplish this lab was to put it in a state machine.
The full state machine flow chart can be seen in Figure 4.

It's a bit big but it's very repetitive. Figure 3 shows a close up of the chart.

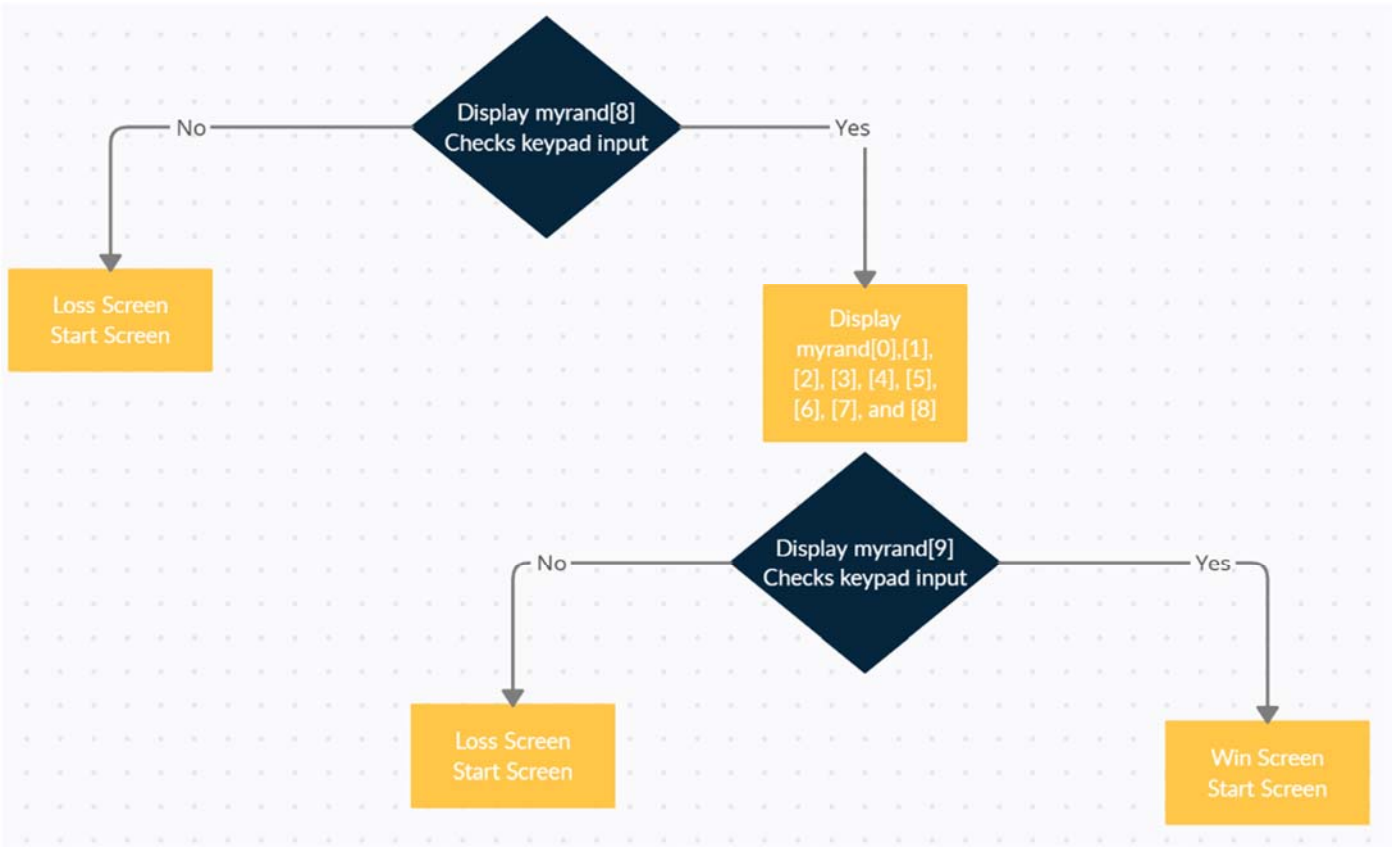

Figure 3: Chart close up

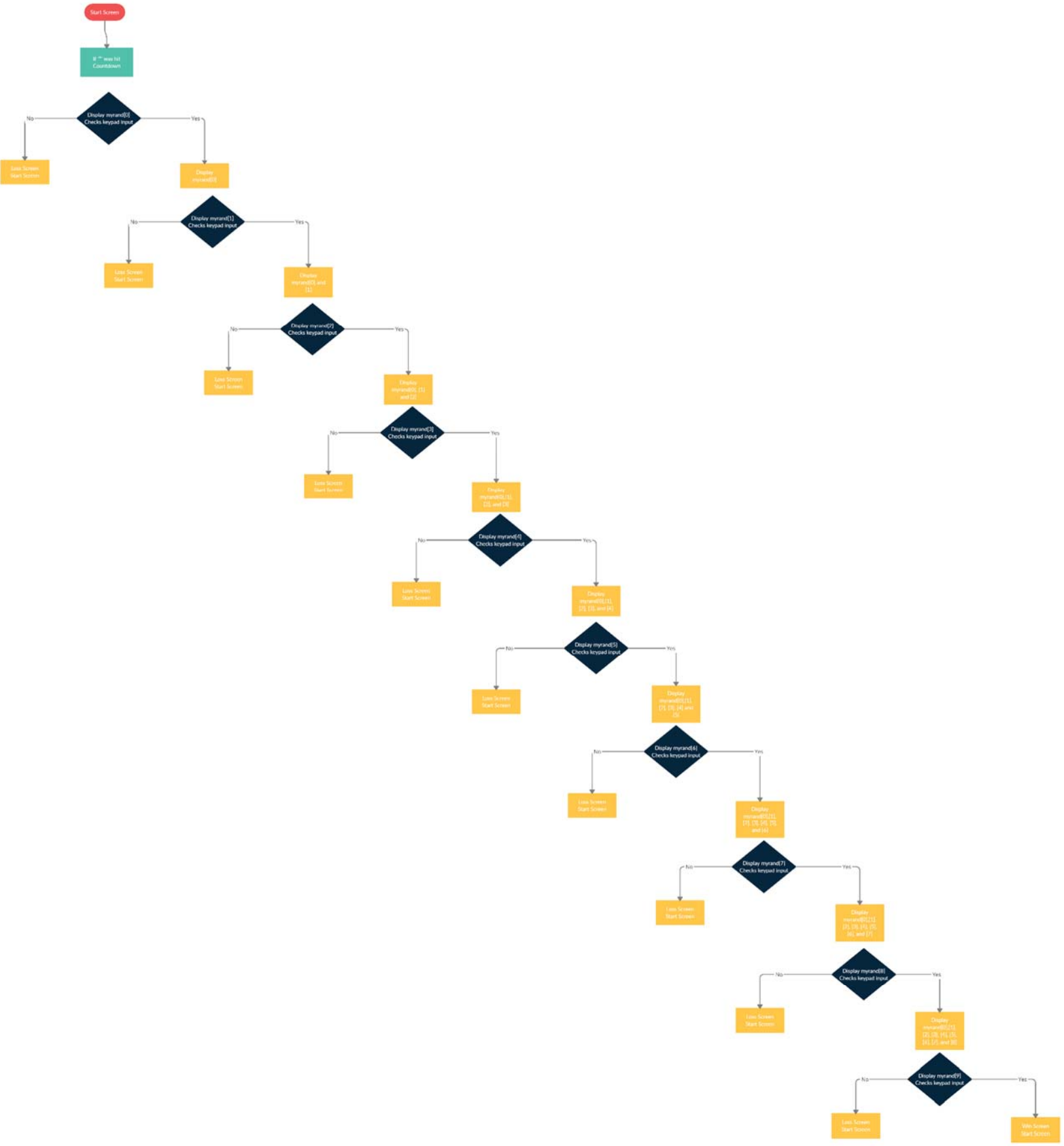

Figure 4: Full flow chart

Getting the input from the keypad and displaying it on the screen took its own function which was drawcurrKey.

It takes in the char that the keypad reads and turns it into numbers on the screen. A number hit other than 1-4 is treated an as game over you lose error.

```
void drawcurrKey(char aChar)
{


    if (aChar == '1'){
        Graphics_drawStringCentered(&g_sContext, "1", AUTO_STRING_LENGTH, 28, 50, TRANSPARENT_TEXT);
        Graphics_flushBuffer(&g_sContext);



    }else if(aChar == '2'){
        Graphics_drawStringCentered(&g_sContext, "2", AUTO_STRING_LENGTH, 48, 50, TRANSPARENT_TEXT);
        Graphics_flushBuffer(&g_sContext);



    }else if(aChar == '3'){
        Graphics_drawStringCentered(&g_sContext, "3", AUTO_STRING_LENGTH, 68, 50, TRANSPARENT_TEXT);
        Graphics_flushBuffer(&g_sContext);




    }else if(aChar == '4'){
        Graphics_drawStringCentered(&g_sContext, "4", AUTO_STRING_LENGTH, 88, 50, TRANSPARENT_TEXT);
        Graphics_flushBuffer(&g_sContext);



    }
}
```

Figure 5: drawcurrKey

Getting the matching value from the keypad to rand was also one of thing I needed to overcome. Since the keypad returns ASCII values, All I needed to do was find the decimal equivalent on the ASCII table to see their equivalence and add 48. This is shown in Figure 6.

```
        break;
    case PLAYING0:
        Graphics_clearDisplay(&g_sContext); //clear
        if (keyPressed == myrand[0] + 48){
            drawcurrKey(keyPressed);
            BuzzerOn();
            P1OUT |= BIT0;       // Set the P1.0 as 1 (High)
            swDelay(2);
            setLeds(0);
            BuzzerOff();
            counter++;
            Graphics_clearDisplay(&g_sContext); //clear
            swDelay(2);
            state = DISPLAY0;
        }else if(keyPressed != 0){

            state = LOSS;
        }
        break;
```

Figure 6: integer to ASCII equivalence

Also using Figure 6: we see the structure of getting the getting the correct number. It will display the number with drawcurrKey, turn the buzzer on, set the LED to high, delay, turn the LED and buzzer off, add 1 to the counter clear display, delay again, and set the state to the next one. The else if is for if the key pressed is not 1-4 and it's and automatic game over.

Once all the road blocks were cleared, I just utilize the state machine and display the numbers and check the keypad for the order of the numbers in the save array of random numbers. Once it reaches the end and all the number are correct, it displays the win screen. After the small delay it reverts back to the start screen of "SIMON". The lose screen is similar. The buzzer is played for a longer period of time and then reverts to the start screen.

Summary and Conclusion:

In conclusion, this lab had the purpose of getting students more familiar with the C programming language and the environment of Code Composer Studio as well as get the hang of using the builder, debugger and the variable window within Eclipse/CCS. This lab makes you build the classic game of Simon by Milton Bradley. This concludes this laboratory experiment with the classic game Simon on the MSP430F5529 using the buzzer, keypad, and Code Composer Studio.

Appendices

ASCII table:
https://canvas.wpi.edu/courses/23392/files/3432901?wrap=1

Lab 1 Handout:
https://canvas.wpi.edu/courses/23392/files/3472073?wrap=1

Demo Project:
https://canvas.wpi.edu/courses/23392/files/3416786?wrap=1

CCS download:
https://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html

CCS tutorial:
https://www.youtube.com/watch?v=2W0ZHO0vzJE

MSP430 user guide:
https://canvas.wpi.edu/courses/23392/files/3432903?wrap=1

C Review pdf
https://canvas.wpi.edu/courses/23392/files/3439859?module_item_id=542026

ECE 2049 Lab BOM:
https://canvas.wpi.edu/courses/23392/files/3411701?module_item_id=527923

ECE 2049 Course Information:
https://www.wpi.edu/academics/calendar-courses/course-descriptions/17856/electrical-computer-engineering#ECE-2049