Phys 319 Formal Lab Report: An Implementation of "Whac-A-Mole" using MSP430 Launchpad

Lang Cheng

Student Number: 40588155

May 15, 2018

Abstract

MSP430 LaunchPad is an effective microcontroller to implement the programs. In this project, "Whac-A-Mole", a popular arcade game, was implemented with this microprocessor using **C**. The player needs to press the correct switches matched to the corresponding LEDs that lighted up. The general functionality of the game was achieved, while there are still some limitations that needs to be improved for a better quality and higher similarity.

Introduction

MSP430 LaunchPad is an open sourced single-board microcontroller which can be programmed by either assembly codes or **C** codes to build digital and iterative objects similarly to the real world, It is an easy-to-use development tool intended for beginners and experienced users to creating microcontroller-based applications. LaunchPads are also equipped with various sets of digital/analog inputs/outputs and can be implemented for a range of sensing and measurement applications.

"Whac-A-Mole" [Fec76] is one of the ever-popular arcade redemption games, which was invented by Aaron Fechter of Creative Engineering, Inc in 1976. In the original game, the player must quickly hit as many moles as possible that would pop out randomly from a target surface with five holes. The personal project was inspired by this game, and it was implemented by MSP430 LaunchPad and programmed in **C**. Similarly, the player needs to push the switches that are matched to the corresponding LEDs which light up in a random order.

Apparatus

- The MSP430 Launchpad
- Wires
- 5 LEDs
- 5 Switches
- 5 1.2$k\Omega$ resistors

As you can see from the circuit diagram below, five LEDs are connected to ports from P1.0 to P1.4 respectively. All those five ports are initialized using MSP430 LaunchPad as the output ports, which can light up the five LEDs based on the random number generated by the codes. In addition, each of the five switches is connected in series with a 1.2k ohm resistor in order to achieve the safety voltage, and they are then connected to ports from P2.0 to P2.4. The player needs to press the correct switches matched to the certain LEDs in order to achieve one score. The current score is shown on the four-digit 7-segment display embedded on the evaluation board.

Display	D3	D2	D1	D0	A1	A0	STR
Launchpad	GND	P1.5	P1.6	P1.7	GND	GND	GND

Figure 1: Connection table for Launchpad and four-digit 7-segment display

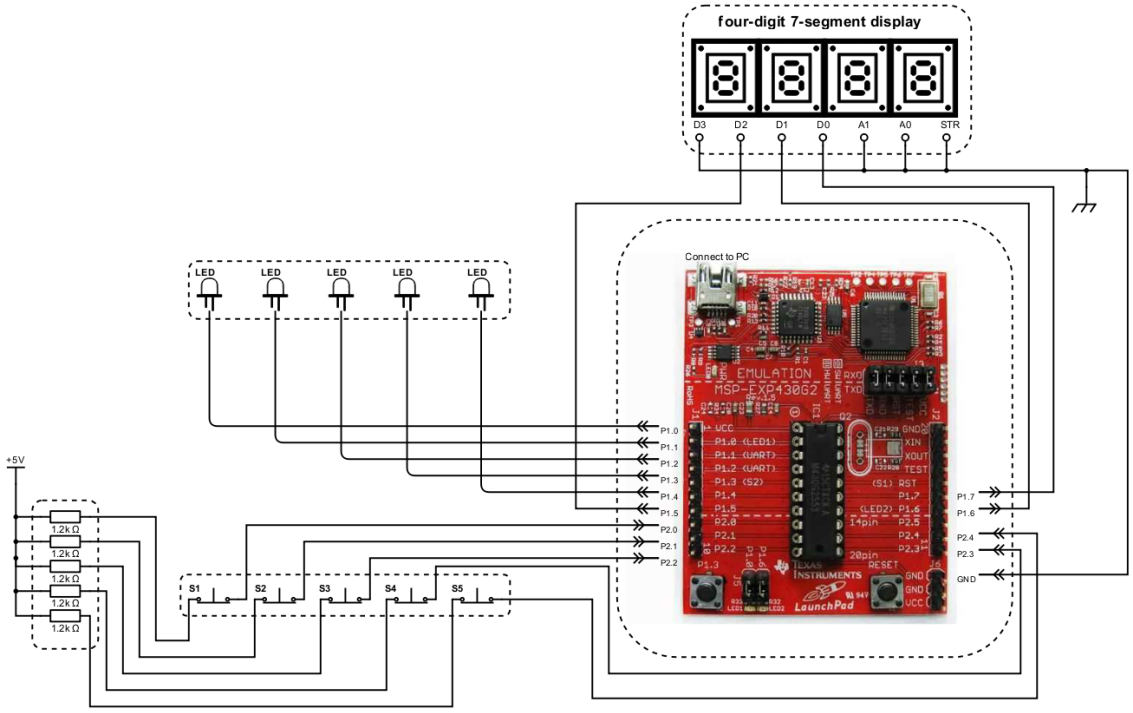

Figure 2: Circuit diagram

Procedure

- After completing the circuit and loading the codes shown in Appendix, the device starts working automatically.
- First, it lights up one or two LEDs depending on the results of the random number generator.
- Then, the player needs to press the correct switches matched to the certain LEDs, otherwise, the program cannot proceed.
- The current score is shown on the four-digit 7-segment display, it increments by one each time the player responses correctly.
- When the player achieves the score of 8, the device keeps blinking the two built-in LEDs which indicates that the player wins the game.
- The “reset” button on the MSP430 board needs to be pressed to restart the game.

Results

The device works smoothly, and it satisfies the expectation and the purpose of this project. However, there are few problems encountered during the progress of this project.

First, the ports available on the board are limited. At the beginning of the project, I planned to set six LEDs and switches, which requires 12 ports in total. However, the four-digit 7-segment display also requires 7 ports in order to fully control the four digits. Therefore, I decreased the number of LEDs and switches to 5 and connected D2, D1 and D0 to the Launchpad, while the rest of them were all grounded. By applying this approach, the scoreboard is able to count up to 8, and all the ports on the LaunchPad can be used efficiently.

Another problem encountered is that MSP430 does not have random number generator. **stdlib.h**[Ltd18a] header file was included in the codes in order to access the standard library of **C**, so function **rand()**[Ltd18b], which is a pseudo-random number generator, can be used directly. However, using this function returns a list of number with the same sequence. In this project, the LEDs were lighted up in the same order every time the "reset" button was pushed. The reason for that is because **rand()** function requires **srand()**[Ltd18c] to seed the random generator function used by the function **rand()**. Also, MSP430 LaunchPad cannot use **srand()** function since the remaining free random-access memory is smaller than the configured stack size after subtracting all used (by either the programs or the library functions) spaces from the MSP's available random-access memory [Gro11]. Therefore, I used a for loop to implement the similar functionality as **srand()**. The loop uses the number generated by **rand()** as an addition to its original statement, and the random number keeps incrementing from 0 to 13 until it reaches the end of the loop. As a result, the loop generates actual random number which differs each time when the "reset" button is pushed.

Discussion

There is no denying that this project has much to be improved. First, the program just pauses when the player does not push the correct switches, this functionality is quite different from that of the original game. So, the built-in timer on MSP430 LaunchPad should be used count the time elapsed, when the value passes the certain threshold, the program automatically proceed to the next round. For example, the player may loss score if he does not response fast enough. It is also recommended to set different levels for this game to measure the skill of the player. For instance, the player can be advanced to the next level with higher difficulty if he successfully hits certain number of moles in a row, such as decreasing the period between each round and increasing the number of maximum LEDs lighted up. It may also be possible to use two MSP430 LaunchPad to interact with each other, so that there are enough ports that can be used. Therefore, the full potential of four-digit 7-segment display can be used to count up to a score of 9999.

For the future project, the LEDs can be replaced with some real moles that are motored by an engine, and those buttons can be set right below the moles. When the player uses a hammer to hit the moles that pop out from the surface, the buttons are also pressed due to the impact force. Then, this project becomes much more similar

to the original game, and the functionality of the program can be well expressed.

Conclusion

In conclusion, the general functionality of "Whac-A-Mole" was achieve using MSP430 and the game was playable, while there are still some limitations that can be improved to achieve a better quality and higher similarity to the original game.

References

[Fec76] Aaron Fechter. *Whac-A-Mole*. Creative Engineering, Inc. Available at:<https://en.wikipedia.org/wiki/Whac-A-Mole>, 1976.

[Gro11] Jens-Michael Gross. *time.h usage (time(null)) fail*. Available at:<https://e2e.ti.com/support/microcontrollers/msp430/f/166/t/110216>, 2011.

[Ltd18a] Tutorials Point (India) Pvt. Ltd. *C Library - <stdlib.h>*. Available at:<https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm>, 2018.

[Ltd18b] Tutorials Point (India) Pvt. Ltd. *C library function - rand()*. Available at:<https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm>, 2018.

[Ltd18c] Tutorials Point (India) Pvt. Ltd. *C library function - srand()*. Available at:<https://www.tutorialspoint.com/c_standard_library/c_function_srand.htm>, 2018.

Appendix

```
#include "msp430.h"
#include <stdlib.h>  // import rand() functions

#define RXD        BIT2
#define TXD        BIT1

unsigned int score;
volatile unsigned int i = 0;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
    DCOCTL = 0;
    // Set DCO to 1 MHz
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    // Initialize the ports that will be used as output
    P1DIR = BIT0 | BIT1 | BIT2 | BIT3 | BIT4 | BIT5 | BIT6 | BIT7;

    // Initialize the ports that will be used as input
    P2DIR &= ~BIT0;
    P2REN |= BIT0;
    P2DIR &= ~BIT1;
    P2REN |= BIT1;
    P2DIR &= ~BIT2;
    P2REN |= BIT2;
    P2DIR &= ~BIT3;
    P2REN |= BIT3;
    P2DIR &= ~BIT4;
    P2REN |= BIT4;




    // Enable interrupts
    __bis_SR_register(GIE);

    // The value that P2OUT needs to be equal to
    unsigned char value;

    // Record the current random number
    unsigned int random;

    while(1)
    {
```

```
// Set P1OUT back to 0
   P1OUT = 0x00;

   // Modify the scoreboard based on the current score value
   if (score == 1) {
       P1OUT |= 0x80;
   } else if (score == 2) {
       P1OUT |= 0x40;
   } else if (score == 3) {
       P1OUT |= 0xC0;
   } else if (score == 4) {
       P1OUT |= 0x20;
   } else if (score == 5) {
       P1OUT |= 0xA0;
   } else if (score == 6) {
       P1OUT |= 0x60;
   } else if (score == 7) {
       P1OUT |= 0xE0;
       // When score == 8, the player wins the game
   } else if (score == 8) {
    // Enter the infinite loop blinking the two built-in LEDs
       for (;;) {
           for (i = 0; i < 5000; i++) {
               if (i == 0 ) {
                   P1OUT ^= 0x01;
               }
               if (i == 2500) {
                   P1OUT ^= 0x40;
               }
           }
       }
   }


   // Insert a delay cycle
   __delay_cycles(500000);

   // Generate a random number
   int add = rand() % 1000;

   // Use a for loop to generate an actual random number
   // The reason to do this will be explained in the report
   for (int j = 0; j < 1000 + add; j++) {
       if (random > 13) {
           random = 0;
       }
       random += 1;
   }
```

```
// Set different P1OUT and value based on the random number
if (random == 0) {
    P1OUT |= 0x01;
    value = 0x01;
} else if (random == 1) {
    P1OUT |= 0x02;
    value = 0x02;
} else if (random == 2) {
    P1OUT |= 0x04;
    value = 0x04;
} else if (random == 3) {
    P1OUT |= 0x08;
    value = 0x08;
} else if (random == 4) {
    P1OUT |= 0x10;
    value = 0x10;
} else if (random == 5) {
    P1OUT |= 0x03;
    value = 0x03;
} else if (random == 6) {
    P1OUT |= 0x05;
    value = 0x05;
} else if (random == 7) {
    P1OUT |= 0x09;
    value = 0x09;
} else if (random == 8) {
    P1OUT |= 0x11;
    value = 0x11;
} else if (random == 9) {
    P1OUT |= 0x06;
    value = 0x06;
} else if (random == 10) {
    P1OUT |= 0xA;
    value = 0xA;
} else if (random == 11) {
    P1OUT |= 0x12;
    value = 0x12;
} else if (random == 12) {
    P1OUT |= 0xC;
    value = 0xC;
} else if (random == 13) {
    P1OUT |= 0x14;
    value = 0x14;
} else if (random == 14) {
    P1OUT |= 0x18;
    value = 0x18;
```

```
        }

        // Waiting for the correct input to proceed the game
        while (P2IN != value);

        while (P2IN == value);

        // Increment the score
        score++;
    }
}
//*******************************************************************************
```