Nate Ambrad and Pranav Jain

ECE2049-C24

31 January 2024

Lab1 – Simon Game Implemented on MSP430F5529

Figure 1 - Custom MSP430F5529 Board displaying Simon welcome screen.

Introduction

The purpose of this lab was to implement the game “Simon Says” on the custom MSP430F5529 board using a program written in Code Composer Studio (CCS). Simon says is a pattern recognition game where a player needs to memorize a pattern sequence of increasing length. The C language was used to write the code for the game. The game makes use of the following board peripherals: LCD, 4 LEDs, Keypad, and Buzzer.

The peripherals are what make the game interactive. The four LEDs on the custom board are lit up in a patterned sequence and the buzzer is used to play a different pitch for each LED. The player is supposed to repeat this sequence by pressing buttons on the keypad which correspond to each LED. The rate at which the pattern is displayed increases with the length of the sequence to add an element of increasing difficulty. The game has a welcome screen with a start button to initiate play and restart button to return to the welcome screen during play. If the player makes an error in repeating the sequence, a lose message appears and the game returns to the welcome screen. If the player successfully completes the full length of the sequence, a win message appears and the game returns to the welcome screen.

The game is written in C as a state machine. Each phase of the game corresponds to a state. There is a state for each of the following procedures: initializing variables, welcome screen, the generated sequence, delay, and player input. The interactions between each state are generally dependent on the user input to the keypad.

Discussion and Results

General Description of Program

The screen and variables are initialized at the start. When the player clicks the start button, a random sequence of 32 numbers is generated that are between 1 and 4. According to the level, the player is played a set of numbers from that randomly generated sequence. Finally, the player is asked to replay the played sequence and if he is wrong, he loses and if he is right, he goes to the next round with 1 extra number to remember. The game keeps becoming faster, making it more difficult for the player to follow the sequence.

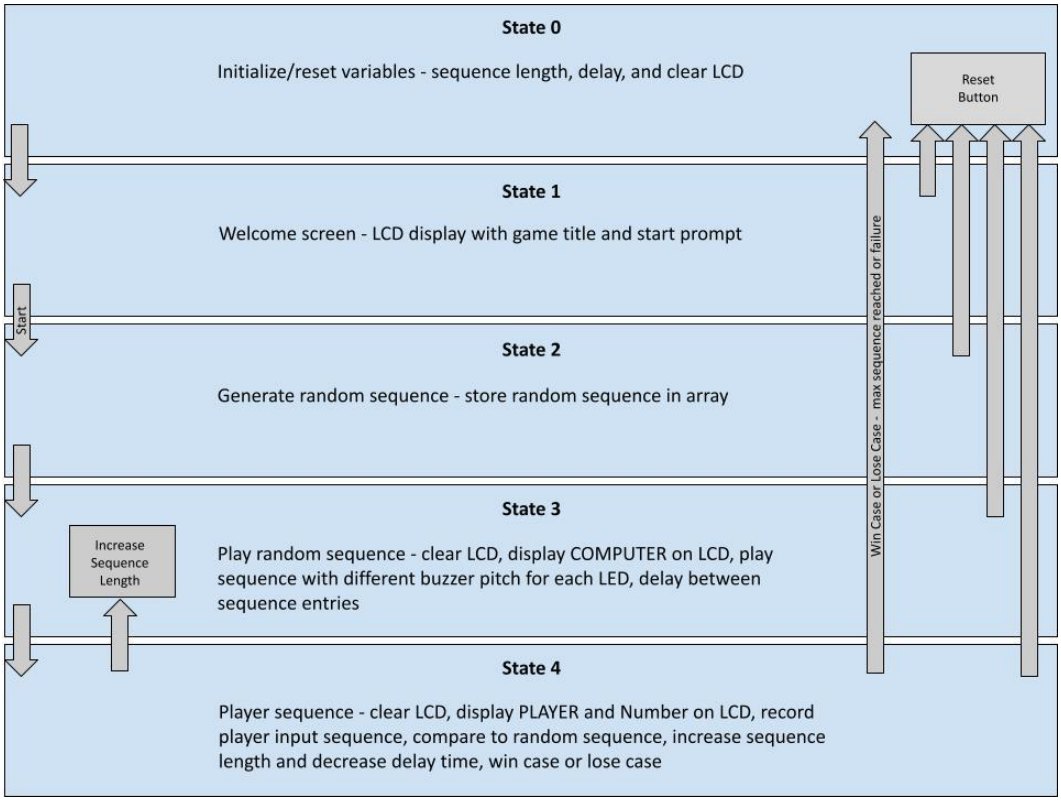

Figure 3 - Flowchart of the state machine for the entire program.

```
#include <msp430.h>
#include <stdio.h>
#include <stdlib.h>
#include "peripherals.h"
```

Figure 2 - C libraries and header files included in main.c

Libraries

<msp430.h> is used to connect with the msp430 and program it.

<stdio.h> is used for programming in the C language.

<stdlib.h> is used for getting the rand() function to create a random sequence.

“peripherals.h” is used for connecting with the keypad and the lights.

Global Variables

```
#define max_seq_length 32          // Sets the maximum length of the sequence to be 32.
#define min_delay 1                // Sets the fastest speed of the sequence to be 1.
#define max_delay 32               // Sets the lowest speed of the sequence to be 32.
unsigned int state = 0;            // Initializes the state to 0.
unsigned int curr_seq_length = 1;  // Sets the current sequence length to be 1.
int comp_seq[max_seq_length];      // Array stores random sequence generated by the computer.
int player_seq[max_seq_length];    // Array that stores the sequence played by the player.
unsigned int curr_seq_index = 0;   // Sets the current sequence index position to 0.
unsigned int curr_delay = max_delay; // Sets the current delay to the slowest speed of the game.
unsigned int currKey;              // Variable to store the value read from the keypad.
int win_flag = 1;                  // 1 or 0, determines whether game is continued or reset
```

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

The maximum length for the sequence provides an endpoint for the game, providing the user an opportunity to have a winning case. Even though it is unlike that a user would reach this point, without the maximum length, the sequence would eventually grow too large to be stored in memory.

Functions

```
// Function Prototypes
void swDelay(char numLoops);
void exit_Simon();
```

Figure 4 - Function prototypes for all non-main functions used in Simon program.

int main(void) is the largest function of the program. Its main purpose is to run the whole program. A switch statement is used and each case acts as a state in the state machine of the program.

void swDelay(char numLoops) is used for adding a software delay by making the computer do random things for a set amount of time.

void exit-Simon() is the reset function. It is used to check if the '#' is pressed to see if the state needs to be changed to 0 and the win_flag to become false.

States 0-4

State 0: The first state initializes all the variables to their starting values. This is really important state, especially when resetting the game using the "#" key.

State 1: The second state types "Simon" on the screen and checks if the "*" key is pressed to start the game.

State 2: The third state is about creating a random 32 sequence of numbers to be used by the computer to play.

State 3: The fourth state is playing the sequence, according to the number of lights that need to light up in that specific round.

State 4: The fifth state allows the player to play the sequence. It also checks if the player won, meaning that he was able to play the 32 number sequence, or if the player lost, meaning that he misplayed a number.

All the states check if the “#” key is pressed in order to reset the game.

Issues and Troubleshooting

Some issues faced included trying to add all the functionality at a single time, making it more complex to debug and find issues within the code. Also, we haven’t used the buzzer before, and it took some time figuring out how to change the tone. Finally, due to the blocking while and for loops present in the code, it took a lot of time to figure out how to include the checkpoints to see if the “#” key was pressed to reset the game. Some bugs such as the game freezing during the sequence or only working at certain sequence lengths occurred but as the program came together those issues were resolved.

Summary and Conclusion

In summary, this lab brought the game "Simon Says" to life on the custom MSP430F5529 board through a CCS program written in the C language. Designed as a pattern recognition challenge, the game utilized board peripherals, including the LCD, 4 LEDs, Keypad, and Buzzer. The LEDs illuminated in a sequential pattern, each accompanied by a unique buzzer pitch. Players replicate the sequence by interacting with the keypad. The game incorporated a welcome screen featuring start and restart buttons as well as an increasing rate of sequence for added difficulty. Knowing what to use the hardware for was not the main challenge. The use of software to control the hardware was. The implementation, structured as a C-based state machine, assigned distinct states to essential procedures such as variable initialization, welcome screen display, sequence generation, delay management, and player input processing. The interactions between these states were intricately dependent on the user's keypad input. The delay management, and input processing proved to be challenging tasks that required multiple different approaches. The implementation of a reset function to control different states required attention as well.

In conclusion, this lab not only successfully achieved the technical implementation of the "Simon Says" game on the custom MSP430F5529 board but also provided experience working on an embedded systems project. The skills of interactive programming and hardware integration were keys to the success of the lab. The careful planning and implementation of the game's mechanics, from the purpose of each peripheral to the state-based control of different procedures, demonstrated the use of software and hardware elements combined to create an immersive, user-friendly program. As a result of this lab, valuable experience was gained and skills with tools such as CCS have been sharpened. This lab was certainly a critical step in the process of learning and applying embedded systems.

Appendices

Prelab – Nate Ambrad

```c
#include <stdio.h>
#include <stdlib.h>
#include <msp430.h>

// Global variables
#define MAX_SEQ_LENGTH 32
#define MIN_DELAY 0
#define MAX_DELAY 10

int main() {
    initLeds();
    configDisplay();
    configKeypad();

    // stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;

    int state = 0;
    unsigned int curr_seq_length = 1;
    unsigned int comp_seq[MAX_SEQ_LENGTH];
    unsigned int player_seq[MAX_SEQ_LENGTH];
    unsigned int curr_delay = MAX_DELAY;

    while(1) {
        switch (state) {
            case 0: // Initializes or resets variables
                curr_seq_length = 1;
                curr_delay = MAX_DELAY;
                state = 1;
                break;
            case 1: // Welcome screen
                Graphics_clearDisplay(&g_sContext); // Clear the display
                Graphics_drawStringCentered(&g_sContext, "SIMON", AUTO_STRING_LENGTH, 48, 15, TRANSPARENT_TEXT);
                // if statement for pressing “*” to start and “#” to reset
                state = 2;
                break;
            case 2: // Generate random sequence and play it
                // Generate sequence
                for (i = 0; i < MAX_SEQ_LENGTH; i++) {
                    comp_seq[i] = rand() % 4 + 1;
                }
                // find way to play it
                // and use buzzer
                state = 3;
                break;
            case 3: // Delay for increasing
                // implement delay between sequence entries
                state = 2;
                break;
            case 4: // Get Player Input and check if it matches computer
                // need to use getKeys() to map button press
                // if player fails to complete sequence (no match)
                state = 0;
                // if player is successful (match)
                state = 2;
                break;
            default:
                state = 0;
                break;
        }
    }
}
```

Figure 5 – Prelab pseudocode with some functional elements mixed in.

Prelab – Pranav Jain

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

unsigned int state = 0;
unsigned int curr_seq_length = 1;
const unsigned int max_seq_length = 32;
unsigned int comp_seq[max_seq_length];
unsigned int player_seq[max_seq_length];
unsigned int curr_seq_index = 0;
const unsigned int min_delay = 0;
const unsigned int max_delay = 100;
unsigned int curr_delay = max_delay;
clock_t start_time;

int main()
{
    initLeds();

    configDisplay();
    configKeypad();

    while(1){
        switch (state) {
            case 0: // Initializes or resets variables
                curr_seq_length = 1;
                curr_delay = max_delay;
                state = 1;
                break;
            case 1: // Display welcome screen
                Graphics_clearDisplay(&g_sContext); // Clear the display
                Graphics_drawStringCentered(&g_sContext, "Welcome", AUTO_STRING_LENGTH, 48, 15, TRANSPARENT_TEXT);
                state = 2;
                break;
            case 2: // Create random sequence
                curr_seq_index = 0;
                while (curr_seq_index < curr_seq_length){
                    comp_seq[curr_seq_index] = rand() % 4 + 1;
                    setLeds(currKey - 0x30); // Plays Sequence
                    curr_seq_index++;
                    start_time = clock();
                    state = 3;
                }
                state = 4;
                break;
            case 3: // Delay
                if (clock() < start_time + (curr_speed * 1000)){
                    state = 2;
                }
                break;
            case 3: // Get Player Input
                curr_seq_index 4 0;
                while (curr_seq_index < curr_seq_length){
                    player_seq[curr_seq_index] = getKey();
                    if ((player_seq[curr_seq_index] >= '1') && (player_seq[curr_seq_index] <= '4')){
                        setLeds(player_seq[curr_seq_index] - 0x30); // Plays Sequence
                        curr_seq_index++;
                        if(player_seq[curr_seq_index] != comp_seq[curr_seq_index]){
                            printf(); // Game ends
                            state = 0;
                        }
                    }
                }
                if (curr_seq_length < max_seq_length){
                    curr_seq_length++;
                }
                if (curr_delay > min_delay){
                    curr_delay--;
                }
                break;
            default:
                state = 0;
                break;
        }
    }
}
```

Figure 6 - Prelab code with most functional elements implemented. Used as starting point for fully developing Simon.

Additional Materials

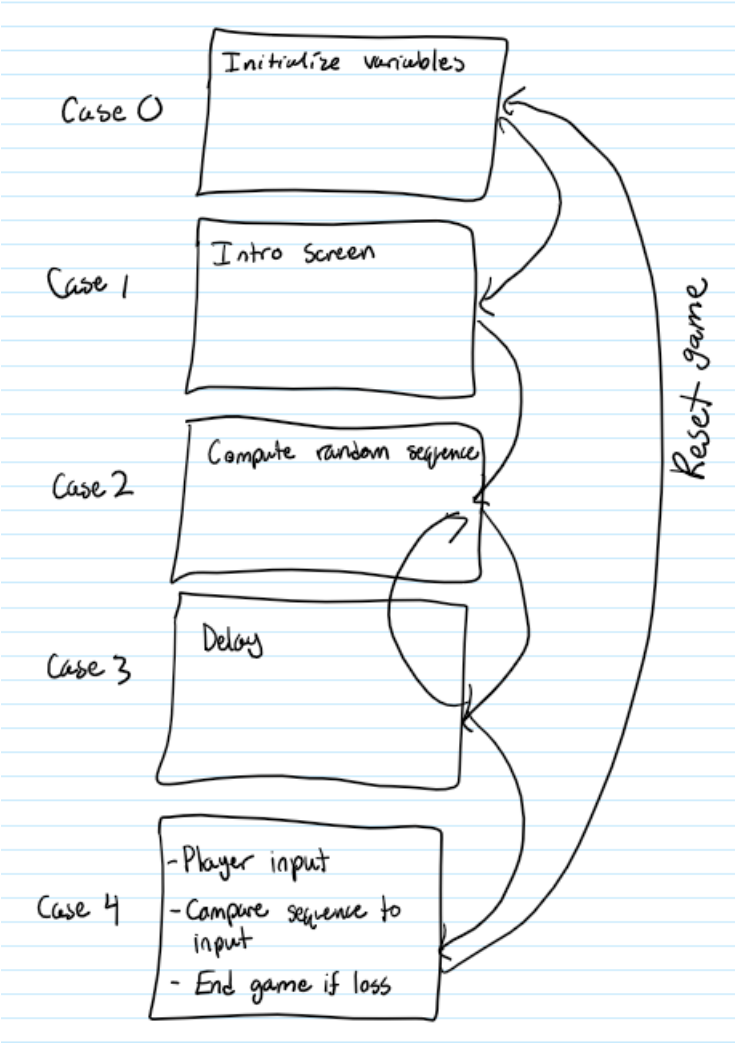

Figure 7 – early development flowchart of the state machine.