LAB REPORT
as a partial requirement for
the course on
FOUNDATIONS OF EMBEDDED SYSTEMS

Lab #1: Implementing a "Simon Says" game

Submitted by:

Submitted to:

Professor Susan Jarvis

February 3rd, 2008

LIST OF FIGURES

INTRODUCTION

The primary purpose of this lab is to create a C program which replicates the "Simon Says" game by Hasbro. In so doing, we also become acquainted with how to use the MSP430 microprocessor and its interface board. The "Simon Says" game created will use four LED lights, as well as, the buttons on the Olimex board as our interface with a user's response. If the user matches the sequence correctly, the speed and sequence of lights increases until failed or victory reached at a sequence of thirty consecutive wins. There exists an array of other features as well, such as sound corresponding with each LED light and a reset key to stop the game – displaying "SIMON". ✓

DISCUSSION AND RESULTS

System Requirements

1) When the game is not being played (or after the 0 key is pressed) the LCD should display SIMON.

   This was easily completed using an IF statement as shown below:

```
getKeys();
if (hitKey =='0')
{
      LEDOff();
      clearLCD();
      writeWord("SIMON");
      swDelay(5);
}
```

✓

2) A new game starts when the # button is pressed

   This was again easily completed, however, we note that this function had to both be implemented in main() to start the game, as well as, while playing Simon Says so that if the # button is pressed during a game, it will still reset. ✓

3) The corresponding LED should flash when the player enters keys (or buttons) 1-4.

This was completed by creating a function called displayLedSeries. This function can be seen in Appendix A as part of the completed program.

4) The sequence should play faster as it gets longer. It is up to the game designer how to implement this. (Hint: Start with a single slow speed and add the variable speed last!)

   This feature was implemented by reducing a delaySpeed counter every 5 rounds. The code snippet of just the counter decrement is shown below:

```
if (i==5 || i==10 || i ==15 || i ==20 || i ==25)
    delaySpeed--;
```

✓

   We note that we could have used % instead of hard coding specific rounds.

5) There is a random number generator function rand() in the C Standard Library (include <stdlib.h>). See the IAR Kickstart help for details on its use.

   A function was created to supply this random number between 1 and 4. It is shown below:

```
int getRandomNum()
{
  return (rand()%4) + 1;
}
```

✓

6) Also realize that you'll need to save the sequence played in order to check it.

   This is vital for game play, and was accomplished by storing each value in an array.

7) Your game should be able to play sequences of up to length 30. Why do you need to select a maximum length for the sequence?

   You need to select a maximum length for the sequence because you must define the array size limit. ✓

8) Any new functions you write should use the port register names defined in msp430x44x.h. No magic numbers!

   This was successfully done using the appropriate BIT# and P#SEL/DIR. The functions configButtons() and getButton() can be seen in Appendix A in the main code.

9) Write a high quality lab report. Your report should include a flow chart of your game (or several flowcharts if that shows functionality better). All flow charts should be computer generated. Include a full code listing in the appendix.

The flow chart of the main program and simonSays function is shown below respectively. These flowcharts includes various extra functionalities that were programmed into our Simon Says game (ie. Decreased delay time).

MAIN PROGRAM FLOWCHART

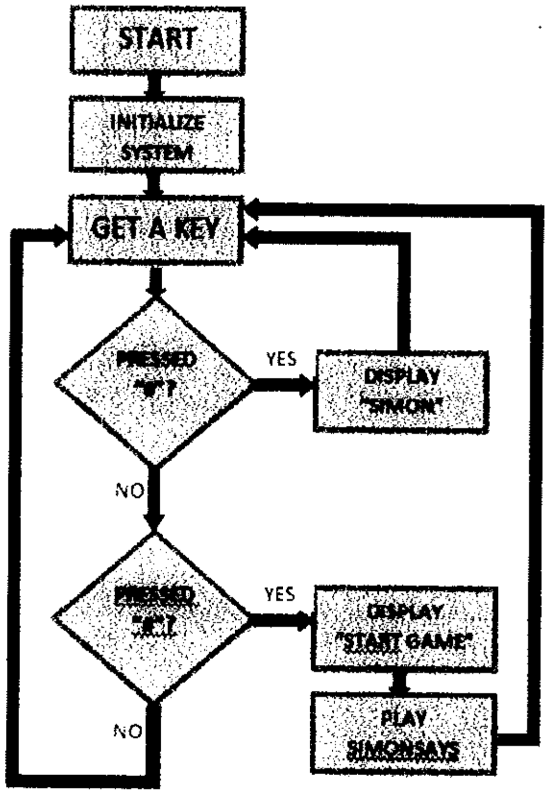
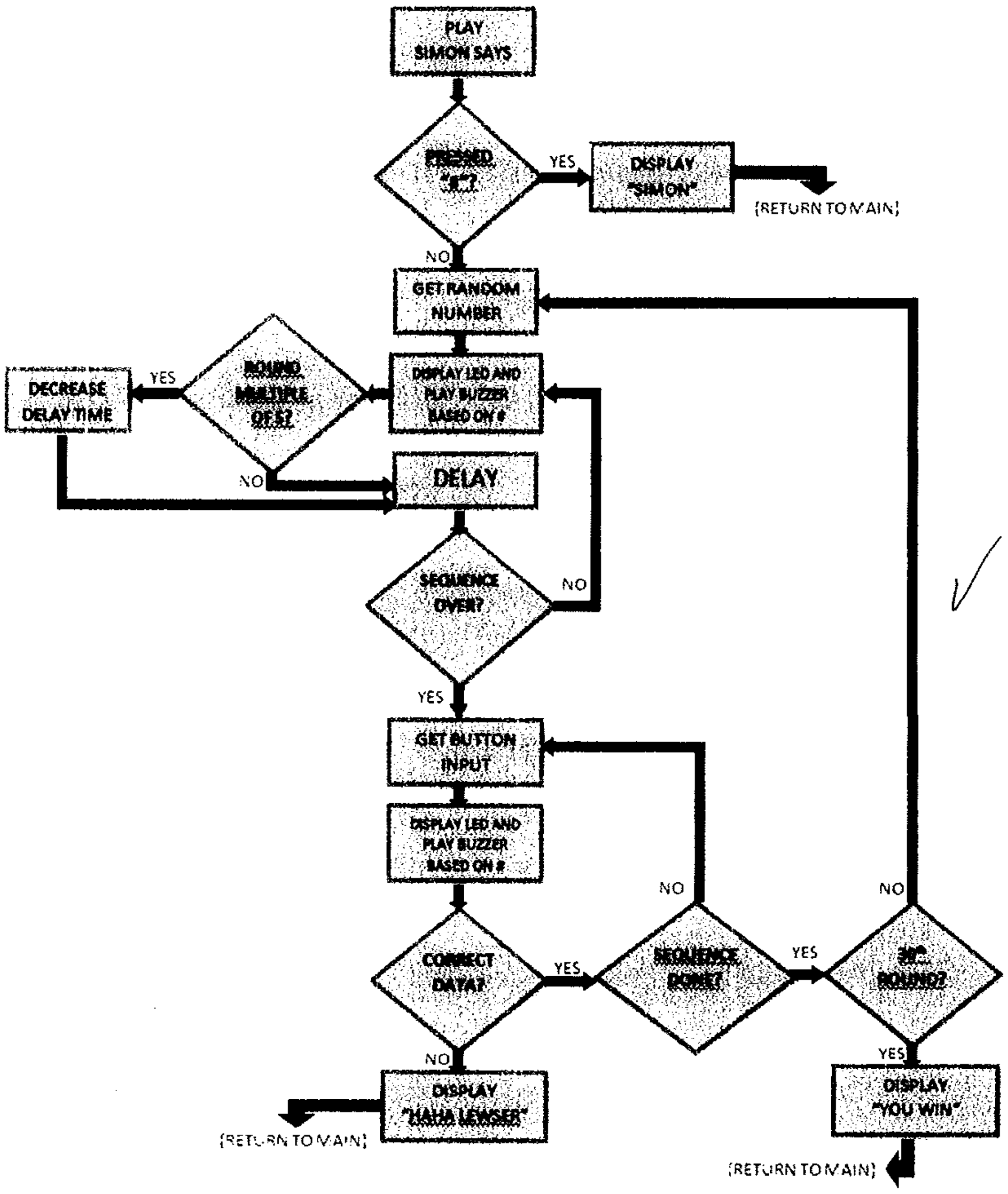


Figure 1 - Main Program Flow Chart

SIMON SAYS FLOWCHART



√

Figure 2 - Simon Says Function Flowchart

CONCLUSION

Success! Our "Simon Says" game functioned fully. Each component worked properly and we were able to increment speed, as well as, use the buttons on the Olimex board to play. While it was a challenge to keep track of all the different functionalities and troubleshoot for bugs, everything worked out in the end. This lab proved very successful in applying the knowledge learned into directly using the MSP430 microprocessor, the IDE, and interface board. ✓

REFERENCES

[1] http://ece.wpi.edu/courses/ece2801smj/msp430-449stk2-a.PDF

[2] http://ece.wpi.edu/msp430/

[3] http://www.freegames.ws/games/kidsgames/simon/simon.htm

[4] http://ece.wpi.edu/courses/ece2801smj/flowchart.pdf ✓

[5] http://ece.wpi.edu/courses/ece2801smj/demo.c

APPENDIX

Appendix A – Complete Simon Says Program

```
// LAB #1 PROGRAM
// BY
// Bas[illegible]                         ation of "Simon Says" game


#include "msp430x44x.h"       // Definitions, constants, etc for msp430F449
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <in430.h>

// ***************************************** FUNCTION DECLARATIONS
**************************************************
void init_sys(void);       // MSP430 Initialization routine

void setupKeypad(void);    // Keypad initialization
void getKeys(void);        // Read from ketpad

void swDelay(unsigned int max_cnt);   // simple SW delay loop

void clearLCD(void);           // Clears LCD memory segments so that LCD is blank
void initLCD(void);            // Setup code to interface LCD with MSP430F449
void writeLetter(int position,char letter); // display single character on LCD
void writeWord(const char *word);     // displays words upto 7 chars on LCD. Can
                                      // also display numbers passed as text
void buzzerOn(int offset);     // turns buzzer on
void buzzerOff(void);   // turns buzzer off
```
✓