Embedded Sequence Controller

Done By: Suha Ahmed

Problem Specification:

This project seeks to build a logic game on a Raspberry Pi, featuring an LCD display for visual output, a button for user input, and LEDs to indicate the game's progress. The system will generate a hidden sequence, compare it to the user's guesses, and use LEDs to provide feedback on the accuracy of the guesses through exact and approximate matches. The game will loop until the user correctly guesses the sequence.

Hardware Specification:

- Raspberry Pi 3 Model B
- 16x2 I2C LCD Display Blue Backlit
- Red LED x1, Green LED x1
- 3 level Potentiometer
- Mechanical Button
- 330Ω resistor x3
- 13 Male-Female, 7 Male-Male

Code Structure:

- mm-matches.s: ARM Assembler function for computing exact and approximate matches.

  Key Functions:

  - initSeq(): Generates a secret sequence of three digits using a loop and the rand function from stdlib.h.
  - showSeq(): Accepts a pointer to a sequence and prints its elements to the terminal using a loop.
  - countMatches(int* seq1, int* seq2): Compares seq2 against seq1 using two loops: one to identify exact matches and another to find approximate matches. It returns a coded value calculated as (exact * 10) + approx.
  - showMatches(code): Decodes the code returned by countMatches() and prints the number of exact and approximate matches.
  - readSeq(int* seq, int val): Converts a numerical input (e.g., 1 2 3) into a sequence by extracting individual digits using basic arithmetic operations.
  - readNum(int max): Parses a guess sequence and stores it in an array called inputs by iterating through the input digits and appending them to the array.

Performance Optimizations:

- Memory Optimization:
    - Dynamic allocation of arrays for sequences using malloc().
    - Proper deallocation using free() to avoid memory leaks.
- Time-Sensitive Operations:
    - Debouncing Button Inputs: Implemented using software delay loops.
    - LED Signaling Timing: Managed via controlled delay functions.
    - LCD Refresh Rate: Optimized to update only when necessary.
- Hardware Constraints:
    - Direct GPIO control using inline assembly for minimal execution overhead.

Hardware Functions:

- digitalWrite(uint32_t *gpio, int pin, int value) – digitalWrite takes 3 arguments, the mapped gpio, the pin to send the value to and the value i.e., HIGH, or LOW. This calculates the pin value and then sends the value to the given value.
    - Calculating offset, mask and gpio register: C
    - Sending value to pin, setting, and clearing pin: Inline Assembly

- pinMode(uint32_t *gpio, int pin, int mode) – pinMode uses the mapped gpio, pin and mode i.e., INPUT, or OUTPUT to configure the given pin as the given mode.
    - Calculating offset, mask and reg: C
    - Setting pin to specified mode: Inline Assembly

- writeLED(uint32_t *gpio, int led, int value) – writeLED uses digitalWrite and pinMode to set the given led pin to be configured as OUTPUT (1) and send the specified value i.e., HIGH (on) or LOW (off) to the given led pin.
    - Uses existing implementations of pinMode and digitalWrite

- readButton(uint32_t *gpio, int button) – readButton uses the mapped gpio and reads when the button has been pressed (i.e., set the value to HIGH), returns 1 when the button is pressed.
    - Calculating offset and register: C
    - Setting pin for button to output is done by using pinMode
    - Reading pin state: Inline Assembly
- waitForButton(uint32_t *gpio, int button) – waitForButton uses readButton and runs an infinite loop to keep checking if the button has been pressed.

Matching function in ARM (mm-matches.s):

Function Name: matches

Inputs:

- A pointer to the memory location storing the secret sequence.
- A pointer to the memory location containing the guessed sequence.

Output:

- Determines how many digits are correct and in the right position (exact matches).
- Identifies digits that are correct but positioned incorrectly (approximate matches).

Implementation Details:

- Iterates through sequences to count exact matches first.
- Then checks for approximate matches while ensuring each peg is counted only once.
- Returns the results as two encoded values within a single register (R0).

Example Output:

Running the game with secret sequence given in as: 3 1 3

(Ignoring all the LCD Put commands such as lcdPutCommand: digitalWrite(25,0) and sendDataCmd(25835032,2)) etc.

project@raspberrypi:~cwk2-sys gcc -o mastermind master-mind.o lcdBinary.o

project@raspberrypi:~cwk2-sys sudo ./master -d -s 313

Printing welcome message on the LCD display ...

Press ENTER to continue:

Round: 1

Turn: 1

Enter a sequence of 3 numbers

Button pressed

Button pressed

Button pressed

Button pressed 2 times

Turn: 2

Enter a sequence of 3 numbers

Button pressed

Button pressed 1 time

Turn: 3

Enter a sequence of 3 numbers

Button pressed

Button pressed

Button pressed

Button pressed 2 times

2 exact

1 approximate

SUCCESS

Summary:

What was achieved:

The embedded system is fully functional, delivering complete gameplay with efficient use of resources through low-level programming techniques.

Outstanding Features:

- Seamless hardware-software interaction.
- Optimized wiring for better efficiency.
- User-friendly interface with LCD feedback for gameplay interaction.
- Simple and intuitive button-based program navigation.

Limitations & Future Work:

- Possibility of adding more flexible game configurations, such as adjustable sequence lengths and a wider range of color options.
- The speed at which the LCD updates could be improved for a smoother gaming experience.

What was learned:

- Gained experience in managing and operating large-scale embedded systems using low-level code.
- Learned how to effectively use peripherals, including buttons, LEDs, and LCDs.
- Developed a better understanding of managing electrical circuits, particularly with components like potentiometers and resistors.