Multiplayer Number Guessing Game Using a PIC18 Microcontroller

Aran Truslove

***Abstract*—An alpha prototype of a multiplayer number guessing game was created using a PIC18F87K22 microcontroller, with code written in PIC18 Assembly language. The aim of the game was to be the first player to guess a random selection of four target numbers. The microcontroller generated four, unique random numbers by reading noise generated by an oscilloscope. Once these numbers were generated, each player was prompted to enter their guess of four numbers through a 4x4 matrix keypad. After each guess, the number of correct numbers in the guess was displayed to the players through a 2x18 character LCD. The game used a total of 2064 bytes of programme memory and 48 bytes of data memory, potentially facilitating the deployment of the software on to the cheaper PIC18F1230 microcontroller.**

I. INTRODUCTION

Historically, there has been a significant engagement with code-breaking games. The board game, Mastermind, has received widespread acclaim since its introduction in the 1970s, selling over 55 million copies [1]. The game involves a code-breaker and a code-maker. The code-maker randomly selects a set of four pegs, with varying colours and places each in slots such that they are not visible to the code-maker. The code-breaker then takes turns to attempt to guess the positions and colours of the four pegs. After each guess the code-maker indicates to the code-breaker the number of correct colours and number of correct positions contained in the code-breaker's guess. Using the information gained after each guess, the code-breaker aims to determine the position and colours of the pegs set by the code-maker before they run out of turns.

The game that has been built, incorporates the code-breaking aspect of Mastermind-style games with a competitive twist. Upon initialisation, four unique, random target numbers, between 1 and 12 are generated by an electronic device. Between one and six players will then take turns to guess these four numbers. The device will display the correct numbers in the guess to all the players. Unlike the Mastermind board game, the positions of the numbers are not considered. The guessing process will repeat until a player guesses all the numbers correctly, after which they will be declared the round winner.

The development of the alpha prototype was guided by the need for precise control and efficiency, which informed our choice of programming language and hardware. The interface was designed to be intuitive, allowing players to receive immediate feedback on their guesses. The random number generation was a core aspect of the gameplay and leveraged external hardware resources to ensure unpredictability.

II. HIGH-LEVEL DESIGN

The number guessing game consists of three main phases: initialisation, player turns and the end of the game. On initialisation, the players are prompted to enter the number of players (between one and six) that will be participating in the round. The prompt is displayed to the players through a 2x16 character LCD and the players input their desired number of players via a 4x4 matrix keypad. Once the number of players has been input, a set of four unique random target numbers, each between 1 and 12 is generated. This is done by sampling a rapidly varying signal generated by an oscilloscope which is converted to a digital value by an analogue-to-digital converter (ADC) and read by the PIC18F87K22 microcontroller.

Once the game round has been initialised with the number of players and the four target numbers, the player turns phase commences with player 1 being prompted to enter their guess. The keypad is used to enter their guess of four numbers, with allowed inputs between 1 and 12 inclusive. Once the player has submitted their guess it is checked against the target numbers and the number of correct numbers in the guess is tallied. If the player does not guess all four target numbers, the number of correct numbers in the guess is displayed to all of the players through the LCD.

When a player successfully guesses all of the target numbers, they will be declared the winner with a message displayed on the LCD. The game will then branch back to initialisation for subsequent rounds to be played. A top-down diagram of the key components of the game design is demonstrated in 1.

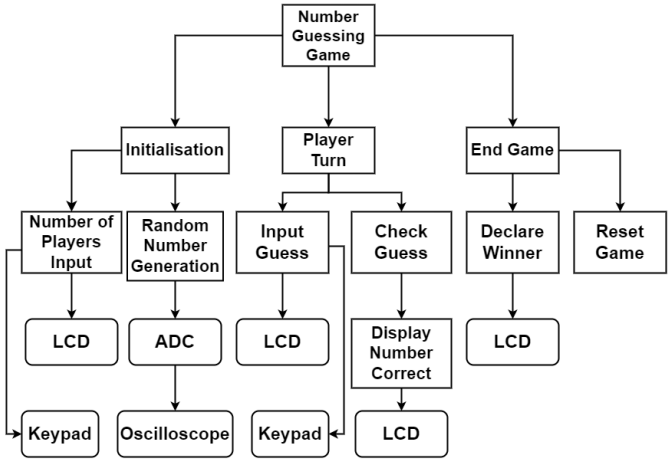

Fig. 1: Top-down diagram of the key software and hardware components used in the number guessing game. Hardware components are indicated by cells with rounded edges.

III. Software and Hardware Design

A. Hardware Design

The PIC18F87K22 microcontroller was used to execute the game logic and coordinate with the peripherals. This microcontroller is equipped with 128 KB of flash programme memory and 4 KB of data memory, which were sufficient to run the game. The EasyPIC PRO v7 board was used during development, providing numerous advantageous features. The board contained the mikroProg programmer and RJ-12 connector, facilitating the programming and debugging of the microcontroller [2]. The numerous ports (A to J) permitted interfacing with external peripherals, including the matrix keypad and digital-to-analogue converters (DAC), providing a bridge to the oscilloscope. The eight-bit TLC7524 DAC provided a direct mapping between the eight-bit ports on the EasyPIC PRO v7 and a breadboard.

1) LCD: The LCD featured 2 rows of 16 cells, where each cell consisted of a 7x5 pixel matrix. The EasyPIC PRO v7 board had an inbuilt connector and interface which supported 2x16 character LCDs in 4-bit mode. Communication with the LCD was done through a CN14 display connector [2].

2) Keypad: The matrix keypad consisted of 16 buttons from 0 to F, as shown in 2. A push switch was located below each button. Each row of the keypad formed a circuit with the low four bits on port E of the EasyPIC PRO v7 while each column formed a circuit with the high four bits on port E. The matrix of push switches, connecting rows and columns is demonstrated in 3.

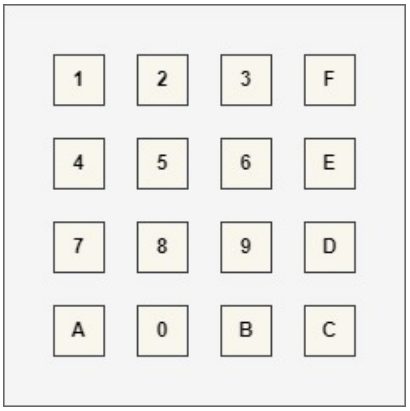

Fig. 2: Block diagram illustrating the layout of the 4x4 matrix keypad used to register player inputs.

To determine which row had been pressed, the low bits of port E were driven to a high voltage while the high bits were driven to a low voltage. Prior to a button being pressed, all the button switches were open. When a button was pressed, the switch corresponding to that button was closed, resulting in the row circuit, originally at a high voltage, being connected to the corresponding low voltage. This resulted in the bit corresponding to that row being driven low hence specifying which button row had been pressed. To determine which column corresponded to the button press, an analogous process occurred, except with the low bits of port E driven high and the high bits of port E driven low. This made it possible

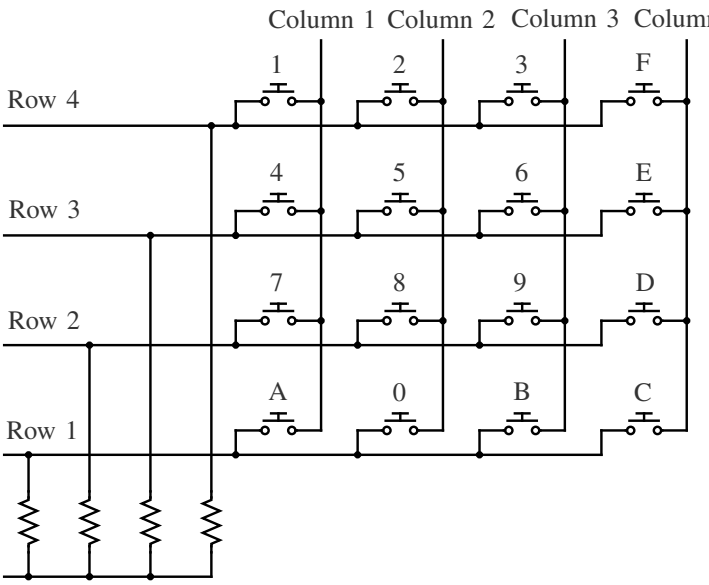

Fig. 3: Circuit diagram demonstrating the matrix of push switches in the keypad without any buttons pressed (all switches open). The bottom left of the diagram illustrates the connection of the lower four bits of port E with the row wires. Adapted from [3].

to determine both the row and column of the button press sequentially, hence specifying which button on the keypad had been pressed.

3) ADC and Oscilloscope: The combination of an ADC and oscilloscope was used to generate the four random target numbers. A sinusoidal wave with a 5 peak-to-peak voltage (V_{pp}) at a frequency of 25 MHz was generated by the oscilloscope, demonstrated in 4.

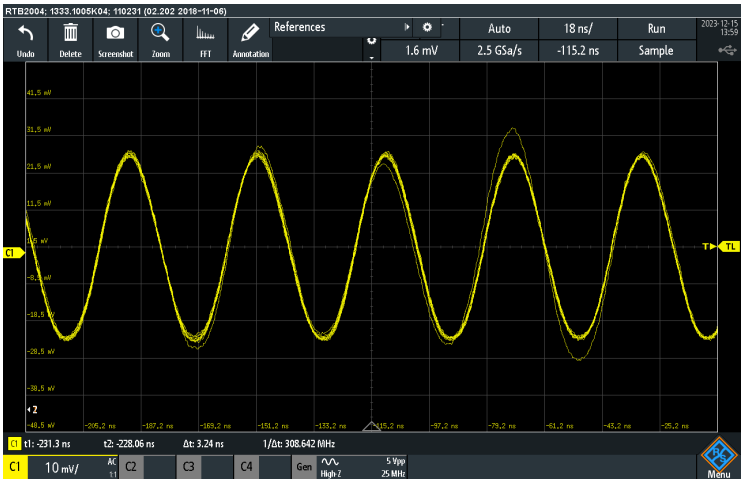

Fig. 4: Oscilloscope screenshot of the 5 V_{pp}, 25 MHz sinusoidal wave used for generating the four random target numbers.

The peak-to-peak voltage and frequency of signal generated were set to the maximum permitted by the oscilloscope to provide the greatest rate of change of voltage with the aim of maximising randomness.

A jumper was used to connect the onboard potentiometer to the RA0 pin on the EasyPIC PRO v7, which is the first pin on port A, enabling the microcontroller to read analogue signals from this pin. The voltage on RA0 was linearly converted to a 12-bit value by the ADC. This meant that a voltage equal to the minimum reference voltage corresponded to a hexadecimal value of `0x0` while a voltage equal to the maximum reference voltage corresponded to `0xFFF`. The ADC reference voltages were specified in the software.

B. Software Design

PIC18 Assembly language was used in the software development of the game. Assembly language can result in a significantly slower speed of development compared to other programming languages, such as C. However, it provides fine-grained control over programme and data memory. This feature is essential when working with highly constrained memory environments.

1) High-Level Game Design: The number guessing game was initialised with the possible permitted keypad inputs, which ranged from '1' to 'E'. The button, 'F', was a special button as it was used as the submit key for guesses and to proceed to the subsequent stages of the game. The `LCD_Setup` subroutine was then called which prepared the LCD to display characters [4]. `Input_Player_Msg` was called followed by `LCD_Write_Message` which prompted an input of the number of players playing (between one and six inclusive). `Character_Input` was called to facilitate the input of characters through the keypad while enforcing restrictions such as a maximum input size and only accepting certain input characters. The maximum number of players was restricted to a single character between one and six inclusive. This input was then stored in memory and the LCD was cleared via the `LCD_Clear` subroutine. Four unique random target numbers were generated with the `RNG` subroutine and stored in memory, ending the initialisation stage of the game.

The player turn was then set to '1' and player 1 was prompted to enter their guess of four numbers. Once they had entered their guess and submitted by pressing 'F', their guess was stored in memory and the LCD was cleared. The `Number_Correct` subroutine was called to determine the number of guess numbers that appeared in the target numbers and the LCD displayed the outcome. The `Press_To_Proceed` subroutine was called to proceed to the next player's turn when 'F' had been pressed, acting as a wrapper around `Character_Input`. If the player did not successfully guess all the target numbers, the next player was prompted to input their guess and this process repeated until a player successfully guessed all of the target numbers.

Once a player guessed all of the target numbers, the programme branched to `end_game` where the current player was declared the winner. Upon pressing 'F' to proceed, the game branched back to initialisation for subsequent rounds to be played. A flowchart summarising the high-level game logic is demonstrated in 5.

2) Keypad Input: The `Keyboard_Press` subroutine was used to wait for a keypad button to be pressed and return the character that had been pressed. This subroutine first called `keyboard_check` which read the lower four bits of port E to determine the button row that had been pressed and then read the upper four bits to determine the button column. 1 ms delays were introduced after updating the input/output status of the bits on port E to ensure that the voltage in the circuit with the keypad had stabilised. The lower four bits and upper four bits were then combined into a single byte by using the `iorwf` instruction. The `keymap` routine was a lookup table that mapped the byte, encoding the row and column information, to the ASCII character on the corresponding button, as shown in 2. If no button was pressed or multiple buttons were pressed simultaneously, then the byte encoding the row and column information would not match up with any of the specified binary patterns and would return the value 0 to indicate this.

To ensure that multiple keypad presses were not registered for a single button press when polling for keypad presses,

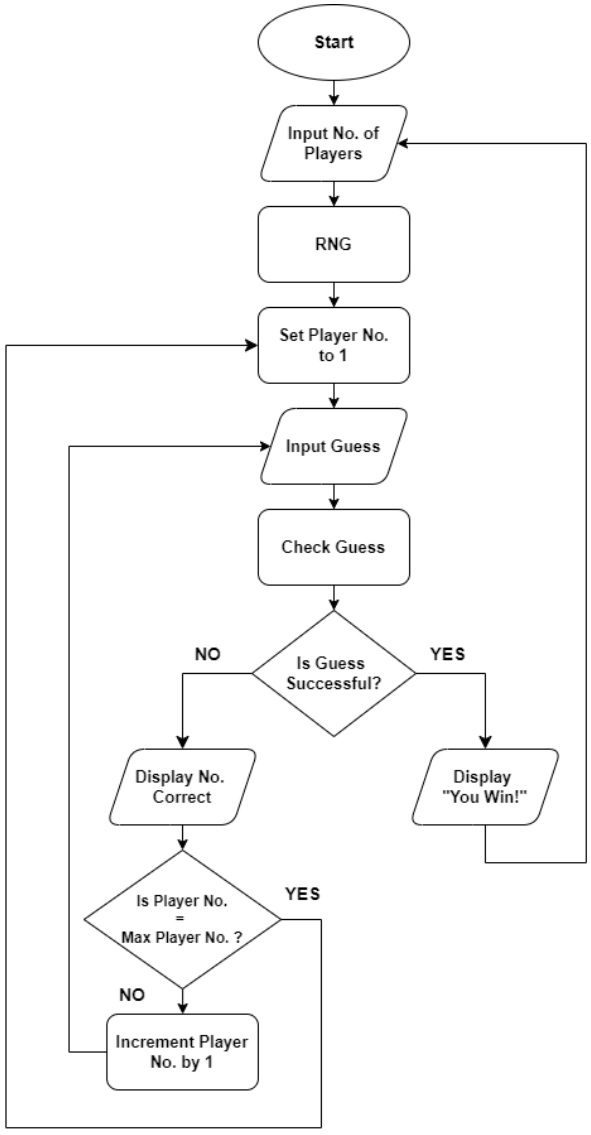

Fig. 5: Flowchart illustrating the high-level logic of the number guessing game, detailing game initialisation, turn-taking, guess checking and progression to subsequent rounds.

Keyboard_Press first waited for keyboard_check to return a value of zero. This indicated that a single key was not already pressed when first entering the subroutine, ensuring that only distinct keypad presses were registered.

The Character_Input subroutine played a crucial role in managing user inputs from the keypad. It accepted four inputs: the memory address of the first permitted character, the number of contiguous valid characters from the memory address of the first permitted character, the maximum number of characters the user was allowed to input and the starting memory address where the user inputs were stored upon submission. The subroutine initially called Keyboard_Press to wait for a keypad press. If the 'F' key was pressed and the required number of valid inputs were met, these inputs were stored in the specified memory block. For other inputs, the character was checked against the set of permitted characters. If the character matched one of the permitted characters, the character was written to the end of the memory block allocated for accepted input characters. The LCD was subsequently cleared and then displayed the characters in the accepted characters memory block before branching back to Keyboard_Press. This cycle was repeated until the maximum number of user input characters had been reached, after which further input characters would not be accepted. A flowchart demonstrating the logic of Character_Input is shown in 6.

3) Random Number Generation: The RNG subroutine was used to generate a set of four unique random numbers, based on binary values converted by the ADC from oscilloscope voltage readings. The subroutine took two inputs: the memory address for the first random number to be written to and the number of random numbers to be generated. For the alpha prototype, the first input was redundant as a constant four random target numbers were generated. However, there was a view of allowing the number of random numbers generated to be chosen by the players in future iterations of the game.

The ADCSetup subroutine was called to configure the ADC by initialising the first pin on port A as an input for reading voltage, with a minimum reference voltage of 0 V and a maximum of 4.096 V [5]. The ADCRead was then used to read the voltage on the first pin of port A and store the corresponding 12-bit binary value in ADRESH:ADRESL. The least significant byte of the binary value was taken from ADRESL and written to WREG. The least significant byte was chosen as these bits would be varying most rapidly with a changing voltage which is favourable when sampling for randomness.

The random eight-bit value contained 256 possible combinations which was far greater than the set of 12 possible characters required. To convert this value to between 1 and 12 inclusive, 12 was iteratively subtracted from the random value until the value was equal to or less than 12. To generate the four unique random numbers, this process was repeated, comparing the current random number to the preceding random numbers. If the random number matched any of the preceding random numbers, then the current random number would be discarded and a new random number would be generated until four unique random numbers were produced.

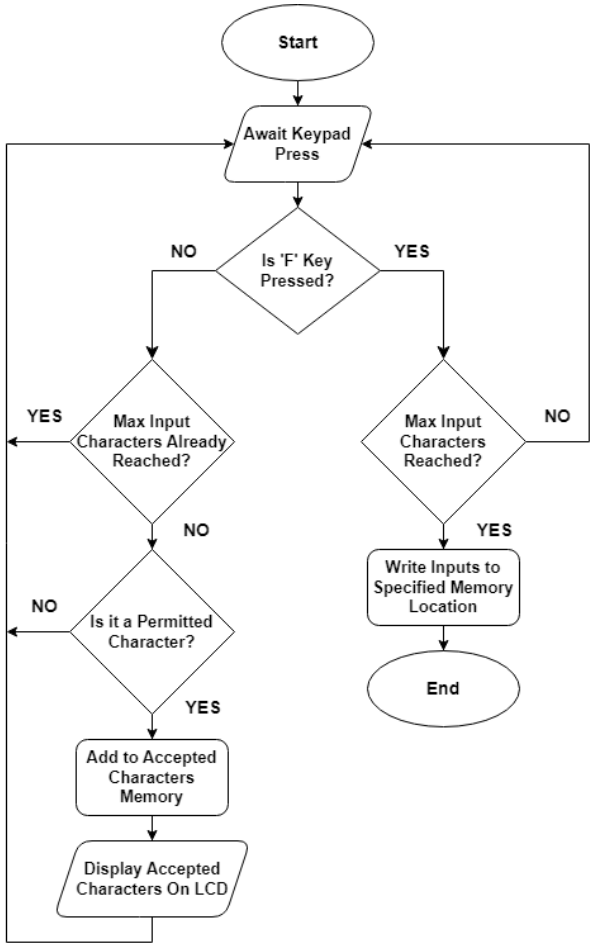

Fig. 6: Flowchart providing a high-level overview of the Character_Input subroutine process for managing keypad entries.

To maintain consistency with the rest of the programme, the random numbers were converted to their corresponding ASCII characters via the Asci_Map subroutine.

4) Guess Evaluation: The Number_Correct subroutine was used to determine the number of correct numbers in a player's guess. It took three inputs: the memory address of the first number in the guess, the memory address of the first number in the target numbers and the number of values to be tested contiguously, which was set to four for the alpha prototype.

Initially, the target numbers were copied to a separate memory block to ensure the immutability of the target numbers during each round. A tally to track the number correct was set to zero. The file select registers, FSR0 and FSR1 were used to traverse the memory spaces of the guess numbers and the target numbers, respectively. The file registers, INDF0 and INDF1 were then used to indirectly reference the values at the memory addresses stored by FSR0 and FSR1.

FSR0 and FSR1 initially stored the first address of the guess and the first address of the target. If the guess and target numbers were equal, the tally was incremented by 1 and the current target value was set to zero to prevent repeat counts. FSR0 was then incremented by 1 and FSR1 set to

the start of the target numbers to recommence the check for the next guess. If the current guess number did not match the target number, then `FSR1` would be incremented to point to the memory address of the next guess number. This process repeated until each guess number was compared to every target number and the tally was subsequently returned in `WREG`.

IV. Performance

The number guessing game performed as intended, free from bugs. Keyboard inputs were correctly validated and appeared on the LCD. Guess evaluation also performed as expected, without displaying contradictory information. Subsequent rounds, with different target numbers, were able to be played consecutively, providing a smooth user experience.

A. Random Numbers

The EasyPIC PRO v7 was equipped with USB-UART communication which allowed data to be streamed from the microcontroller to external devices. This feature was used to collect data on the random numbers generated. 10K sets of random target numbers were recorded for the case of random numbers being sampled from the oscilloscope and when port A was not connected to the oscilloscope.

A Chi-square test at the 0.05 significance level was used to test the randomness of the numbers generated. The null hypothesis, H_0, was taken to be the case where the number of occurrences of each of the random numbers was uniformly distributed while the alternative hypothesis, H_1, was the case where the number of occurrences was not uniformly distributed. For the set of random numbers generated with the oscilloscope, the Chi-square test results in a p-value of 4.3×10^{-17} and a mean percentage difference of the number of occurrences from a uniform distribution of 4%. Although H_0 was rejected and H_1 was accepted at the 0.05 significance level implying that the generated numbers were not random, the mean deviation of 4% is likely inconsequential for the practical purposes of the game.

Another identical Chi-square test was carried out for data corresponding to the case when port A was not connected to the oscilloscope. The p-value for this test was very small and less than the smallest positive number representable in the floating-point precision used by Python, resulting in H_0 being rejected. The mean deviation of the number of occurrences with the expected number for a uniform distribution was 35%. This deviation is significantly greater than for the random numbers generated with the oscilloscope demonstrating the necessity for external sources of randomness in the alpha prototype.

B. Memory Efficiency

The map file, which was generated during the build of the programme, was used to analyse the memory usage. This file indicated that 2046 bytes of programme memory and 48 bytes of data memory were utilised by the software. This corresponded to 1.6% of the programme memory and 1.2% of the data memory capacity of the PIC18F87K22 microcontroller. As this represented a very low usage of the hardware memory resources, the usage of other PIC18 microprocessors was explored. The PIC18F1230 microcontroller is equipped with 4096 bytes of programme memory and 256 bytes of data memory, providing sufficient capacity to run the software with some margin for additional features. This microcontroller costs £2.98 [6] which is 35% cheaper than the PIC18F87K22, costing £4.61 per unit [7]. This cost reduction would provide a significant advantage if the game were to be sold commercially.

V. Modifications and Improvements

Although the core features of the alpha prototype performed as intended, modifications should be implemented to ensure that the beta prototype is more commercially viable. The first issue to address is the generation of random numbers. An oscilloscope is a larger and more expensive piece of kit relative to the other components so it should not be relied upon as a source of randomness. Ideally, the microcontroller should be able to source random noise from the surrounding air. An approach to improve the randomness of this method would be to decrease the range of the reference voltages on the ADC. This would have the effect of amplifying smaller variations in voltage since a smaller voltage range would be mapped to the same 12-bit possibilities of random numbers. This should help to improve the randomness of the target numbers generated without relying on external sources of randomness.

Further enhancements to the memory efficiency and software maintainability could be made. There were two copies of identical delay subroutines situated in the `keyboard.s` and `LCD.s` files. This could be improved by transferring the delay subroutines to an external file such as `utils.s` and calling the delays from this file. The delay subroutines occupy 42 bytes of programme memory so eliminating the copy will reduce the programme memory utilisation from 2064 bytes to 2022 bytes.

To improve the user visual experience, a 128x64 graphic LCD (GLCD) could be used in place of the 2x18 character LCD. As with the 2x18 character LCD, there is a slot allocated for the 128x64 GLCD on the EasyPIC PRO v7 ensuring a smooth development experience but a disadvantage is that each pixel has to be specified rather than simply transmitting ASCII code characters which may increase memory consumption. A potential addition to the game would be to add a countdown for each player's turn, introducing an increased level of challenge and urgency. This feature could be implemented by leveraging built-in timers and interrupt capabilities on the PIC18 microcontroller. These tools can be programmed to trigger specific instructions based on elapsed time, facilitating the countdown functionality.

VI. Conclusion

An alpha prototype of a multiplayer number guessing game was successfully developed using a PIC18F87K22 microcontroller and programmed in PIC18 Assembly language, achieving efficient memory usage. The game integrated a 2x16 character LCD, 4x4 matrix keypad, oscilloscope and

ADC to provide an interactive user experience. The primary features of the game functioned as intended and the low memory usage potentially facilitates the use of the cheaper PIC18F1230 microcontroller in future iterations. The target number generation was found to be more random when sampling from an oscilloscope voltage rather than directly sampling noise on port A, unattached to an oscilloscope. To improve the randomness when directly sampling from port A, decreasing the range of the reference voltages was suggested. Additionally, improvements to memory efficiency could be made by consolidating redundant delay code and upgrading to a GLCD could offer a higher resolution, potentially enhancing user experience.

VII. Product Specifications

- **Microcontroller: PIC18F87K22**
  - Data Memory: 4 KB
  - Programme Memory: 128 KB
- **Development Board: EasyPIC PRO v7**
  - Compatible Microcontroller: PIC18F87K22
  - Connectivity: Ports A-J, UART, CN14 display connector
  - Debugging Features: mikroProg supporting real-time debugging
  - Other Features: 12-bit ADC
- **Display: 2x16 Character LCD**
  - Cell Resolution: 7x5 pixels
- **User Input: 4x4 Matrix Keypad**
- **Oscilloscope: Rohde & Schwarz RTB2004**
  - Features: Waveform Generator
- **Software**
  - GitHub: https://github.com/arantruslove/Multiplayer-Mastermind/tree/lab-testing

References

[1] (2000) A brief history of the master mind boardgame. Accessed: 26/11/2023. [Online]. Available: https://web.archive.org/web/20150906044819/http://www.tnelson.demon.co.uk/mastermind/history.html

[2] *EasyPIC PROv7 User's Guide*, MikroElektronika, [Online; Accessed: 06/01/2024. [Online]. Available: https://www.mikroe.com/easypic-pro-v7

[3] M. Neil, "Decoding and using a 4x4 keyboard," Presentation slides in Imperial College London, Physics Microprocessors Laboratory, [Accessed: 06/01/2024].

[4] pptman, "Imperial college london/microprocessors lab," 2021, [Online; Accessed: 06/01/2024]. [Online]. Available: https://github.com/ImperialCollegeLondon/MicroprocessorsLab/tree/LCD_Hello_World

[5] M. Neil, "Device drivers - measuring voltages," Presentation slides in Imperial College London, Physics Microprocessors Laboratory, [Accessed: 07/01/2024].

[6] Microchip Technology Inc., "Pic18f1230 product information," Online, Accessed: 06/01/2024. [Online]. Available: https://www.microchip.com/en-us/product/PIC18F1230

[7] Microchip Technology Inc, "Pic18f87k22, product information," Online, Accessed: 06/01/2024. [Online]. Available: https://www.microchip.com/en-us/product/PIC18F87K22