Homework 9: Software Design Considerations

Team Code Name: _____Treasure Chess_____ Group No. __2__

Team Member Completing This Homework: __Parul Schroff_____

E-mail Address of Team Member: pschroff @ purdue.edu

Evaluation:

SEC	DESCRIPTION	MAX	SCORE
1.0	Introduction	5	
2.0	Software Design Considerations	30	
3.0	Software Design Narrative	30	
4.0	Summary	5	
5.0	List of References	10	
App A	Flowchart/Pseudo-code for Main Program	10	
App B	Hierarchical Block Diagram of Code Organization	10	
	TOTAL	100	

Comments:

Comments from the grader will be inserted here.

1.0 Introduction

Our project is a voice-controlled chess game that uses two 16x32 RGB LED matrix panels to display the chess board. A player can input the moves to play through a microphone or a keypad. Thereafter, the game logic checks if the move played is correct or not in two ways: if the square the piece is being moved to is empty or occupied by the opponent's piece, and if the move played by the respective piece is correct – for example rook only travels horizontally or vertically, bishop only travels diagonally, etc. Once the game logic has assessed the move, it then sends the output to the RGB LED matrix panel to display the current state of the chessboard. Our design also has provision for displaying the possible moves for a piece selected by the player to move. Additionally, we will be using OLEDs to display time and other in-game statistics during the game.

2.0 Software Design Considerations

The main microcontroller of our design PIC24EP512GU810 has on-chip flash memory of 512 KB. For the purpose of this document, we will refer to this microcontroller as PICEP. We will use the Run-Time Self-Programming method to program the flash memory which allows the user application to modify Flash program memory contents [1]. Most of this memory is utilized for game logic. The program memory map for the PICEP can be seen in Figure 3 (Appendix C). User Program Flash Memory is restricted to the address range 0x000200 to 0x0557FE. Data storage for our design is limited to the current state of the chessboard, any pieces lost during the game, total number of moves played, status of pawns and status of rooks and kings (for implementing en passant capturing and castling respectively), corresponding color of player 1 and 2 and the checkmate status that checks if the game has reached to an end or not. All of these are extern global variables that are stored in the data segment of the program memory [12]. The executable instructions for program are stored in the text segment. The voice recognition microcontroller dsPIC33FJ128GP202 has an on-chip flash memory of 128 KB which is used for packing the inputs (1-8 and a-h) for the coordinates of the square of the piece received through the microphone. For the purpose of this document, we will refer to this microcontroller as dsPIC. For both the microcontrollers, we are using the RPn/RPIn pins for any of the remappable peripherals.

We are using the MPLAB Integrated Development Environment for programming and debugging in C for both the microcontrollers [2]. MPLAB ICD 3 In-Circuit Debugger probe is connected to the PC using a high-speed USB 2.0 interface and to the microcontroller development board using an RJ-11 header. Microchip C 30 Toolsuite provides the MAPLAB C30 C Compiler which uses pic30-gcc to compile C code used for the project.

Voice input from the microphone will first get transmitted to the Si3000 voice band codec [3] which will perform the analog to digital conversion. Each of the input goes through a mixer prior to ADC conversion. Subsequently, data is sent to the dsPIC for completing the voice recognition and packing in the inputs received.

Transmission between the two microcontrollers is accomplished by using the UART channel [4]. UART will use the UxCTS (clear to send) and UxRTS (request to send) hardware controlled pins. This will allow it to operate in flow control mode. This works similar to the handshake protocol. Unless the PICEP is ready to receive another input from the dsPIC, data will not be transmitted between the two microcontrollers. UART will run at the standard speed mode that is, 16x baud clock. For the UART mode register, UARTEN, UEN <1:0> and BRGH will be used and ABAUD (Auto-Baud) will be disabled. Data bits <7:0> will be accordingly used for the UART Receive and Transmit register to transfer the characters. UART Baud Rate Generator Register <15:0> will store the value such that the baud rate value is 9600 at an oscillating frequency of 7.37 MHz

The Type A Timer in asynchronous counter mode for the PICEP will be used timing the game [5]. Register T1CON controls the Type A Timer. Bit 15, TON, either starts or stops the timer; TSYNC is set to 0 to disable external clock input synchronization and Bit 1, TCS is set to 1 to enable input from external clock; Bits 5-4, TCKPS <1:0> are set to 00 for a prescale value of 1:1. TMR1, timer count register (initialized to 0) and PR1, period register (set to 32767) are also used with the required values.

Output from the PICEP to the OLEDs is sent through the SPI module. SPI is used under the Master mode of operation as the master (PICEP) will just be outputting the data to the OLED (slave) [6]. 8-bits are transferred at a time as the shift register being used in our design is an 8-bit parallel out shift register [7], hence, for SPIxCON1 register, MODE16 is set to 0. Bit 11 of the same register, SDOx (serial data output pin), is controlled by the module therefore DISSDP is set to 0. Bits CKE, SMP and CKP are also set to 0 under the Master mode. MSTEN is set to 1 as we

are running the module under Master mode and setting SPIEN to 1, enables the SPI module. Register SPIxCON2 is set to 0 as we are not performing framed SPI operation.

In addition to voice input, a player can also manually send the moves to play through 16-key keypad. Input from the keypad directly goes to the PICEP microcontroller. For scanning the keypad, we will use the sequential exploration of rows method [8]. In this method, all the rows are configured as output and the columns as inputs to the microcontroller or vice versa. Default input for all the column lines is set to 1. The keypad is then scanned by setting the rows to 0 one at a time and reading the columns. This method even though scans 16 keys, only requires 8 pins on the micro (Figure 5, Appendix C) making it more efficient.

PWM channels for the display are used to increase the speed of refreshing the RGB LED Matrix panel [9]. It will work under the independent output mode (PMOD=11). GPIO Pins A, B and C are used to control which rows the data will go to.

Overall organization of the application code is command driven (Application Code 3) as our design depends on the user input and also runs through the same code every time (Figure 1, Appendix A). Unless the player has provided any input, we do not want the game to run. Once the move is entered, the program checks if it is legal and/or displays the possible moves for the selected piece. Once the move is correctly assessed, chessboard is displayed on the RGB LED matrix panel. Therefore, even though, same code is implemented, running the program continuously in an infinite loop is not required. The game and hence the executable code end when either of the kings has been checkmated.

3.0 Software Design Narrative

Figure 1 in Appendix B depicts the hierarchical view of the various code modules. Chessboard layout on the RGB LED matrix is shown in Figure 6 (Appendix C). The rows (ranks) are numbered 1 through 8 bottom-up, and the columns from ‘a’ through ‘h’, left to right. White pieces are placed on rank 1 and 2 (bottom two rows) and the black pieces are placed on rank 7 and 8 (first two rows). Upon start-up, input is received from either the microphone or the keypad according to user preference. The player will speak out or press the coordinates of the current position of the piece and that of the next position they want to move to. The input from the microphone will be transmitted to the PICEP (game and display logic microcontroller) after passing through the Si3000 voice band codec and the dsPIC (Section 2).

The PICEP performs the game logic and sends the correct output to the RGB LED matrix panel. Once the coordinates are received from the player, the game logic begins its magic. Most of the functions to check if the move played by the player is legal or not are under one source file. At first, the piece that occupies the current position as provided by the player is retrieved. Then the square to which the player wishes to move is checked if it is empty or occupied by an opponent piece. Then for each piece, we check if the corresponding move is possible or not. For example: a rook can only move horizontally or vertically, a bishop can move diagonally, queen can move either like a rook or a bishop, king can move to any of the 8 surrounding squares or engage in castling in conjunction with a rook provided it does not result in the king being in a check state, a knight can 'jump' in an L-shape; rules for en passant capturing for pawns is also taken into consideration [10]. Our game also has provision to display the possible moves for a piece selected by the player to play [11]. Whether the possible moves will be displayed each time a player makes a move, or it will be an option using a pushbutton, is still undecided. After the move is ascertained to be legal, the game logic will reset the current state of the chessboard and send the output to the RGB LED Matrix.

For displaying various pieces on the RGB LED Matrix panel, each piece utilizes 4x4 RGB LEDs of the two 16x32 matrices. Having RGB LEDs gives us the freedom to pick any color for the players and the squares; exact colors being used still need to be determined. For each piece, we will display it using a corresponding letter: C for rook, H for knight, B for bishop, K for king, Q for queen and P for pawn (Figure 4, Appendix C). Our design will also keep track of the number of pieces lost during the game and the time elapsed since the game has started. Both of these statistics will be displayed on the OLED screens as an output. This provision is simply for the convenience of the players.

Almost all of the game logic has been successfully coded and tested in pure C, that is, it compiles using gcc. Interfacing it with the PICEP microcontroller is the next step which might require minor changes syntactically; code for voice recognition and output display still needs to be written and tested.

4.0 Summary

Our project is a voice controlled chess game that takes the input from a microphone or a keypad and uses the RGB LED matrix panel to display the chess board. Our design not only checks if the moves played by each player are legal or not, but also displays possible moves for each selected piece. Additionally, we are using OLEDs to output various in-game statistics. Transmission between the two microcontrollers is handled by the UART protocol. The game and display logic microcontroller (PIC24EP512GU810) sends the data to the OLEDs using the SPI module and uses the PWM for fast refreshing of the RGB LED matrix. Our overall organization of the application code is command driven. Even though we repeat the same instructions after a move is played, the game does not progress unless the player has played correct moves.

5.0 List of References

[1] "Section 5. Flash Programming," Microchip, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70609D.pdf. [Accessed 21 March 2013].

[2] "MPLAB ICD 3 In-Circuit Debugger," Microchip, [Online]. Available: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en537580. [Accessed 21 March 2013].

[3] "Si3000 Voice Band Codec with Microphone/Speaker Drive," Silicon Labs, [Online]. Available: https://www.silabs.com/Support%20Documents/TechnicalDocs/si3000.pdf. [Accessed 21 March 2013].

[4] "Section 17. UART," Microchip, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70582D.pdf. [Accessed 21 March 2013].

[5] "Section 11. Timers," Microchip, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/S11.pdf. [Accessed 21 March 2013].

[6] "Section 18. Serial Peripheral Interface (SPI)," Microchip, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70569C.pdf. [Accessed 21 March 2013].

[7] "SN54LV164A, SN74LV164A 8-Bit Parallel-Out Shift Registers," April 2005. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn74lv164a.pdf. [Accessed 21 March 2013].

[8] "Lab 18: Matrix keypad interfacing," Embedded Lab, 25 August 2011. [Online]. Available: http://embedded-lab.com/blog/?p=3428. [Accessed 21 March 2013].

[9] "Section 14. High-Speed PWM," Microchip, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70645C.pdf. [Accessed 21 March 2013].

[10] P. Schroff, "Check If Legal," Treasure Chess, March 2013. [Online]. Available: https://engineering.purdue.edu/477grp2/nb/Game%20Logic/checkIfLegal.c. [Accessed 21 March 2013].

[11] P. Schroff, "Possible Moves," Treasure Chess, March 2013. [Online]. Available: https://engineering.purdue.edu/477grp2/nb/Game%20Logic/possibleMoves.c. [Accessed 21 March 2013].

[12] P. Schroff, "Main Declarations," Treasure Chess, March 2013. [Online]. Available: https://engineering.purdue.edu/477grp2/nb/Game%20Logic/mainDeclarations.h. [Accessed 21 March 2013].

[13] "16-Bit Microcontrollers and Digital Signal Controllers with High-Speed PWM, USB and Advanced Analog," Microchip, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70616g.pdf. [Accessed 21 March 2013].

[14] "Keypad Scan," [Online]. Available: http://esd.cs.ucr.edu/labs/decode_key/decode_key.html. [Accessed 21 March 2013].

[15] E. Scimia, "How to Set Up a Chess Board," About.com, [Online]. Available: http://chess.about.com/od/rulesofchess/ss/Boardsetup_7.htm. [Accessed 21 March 2013].

Appendix A: Flowchart/Pseudo-code for Main Program

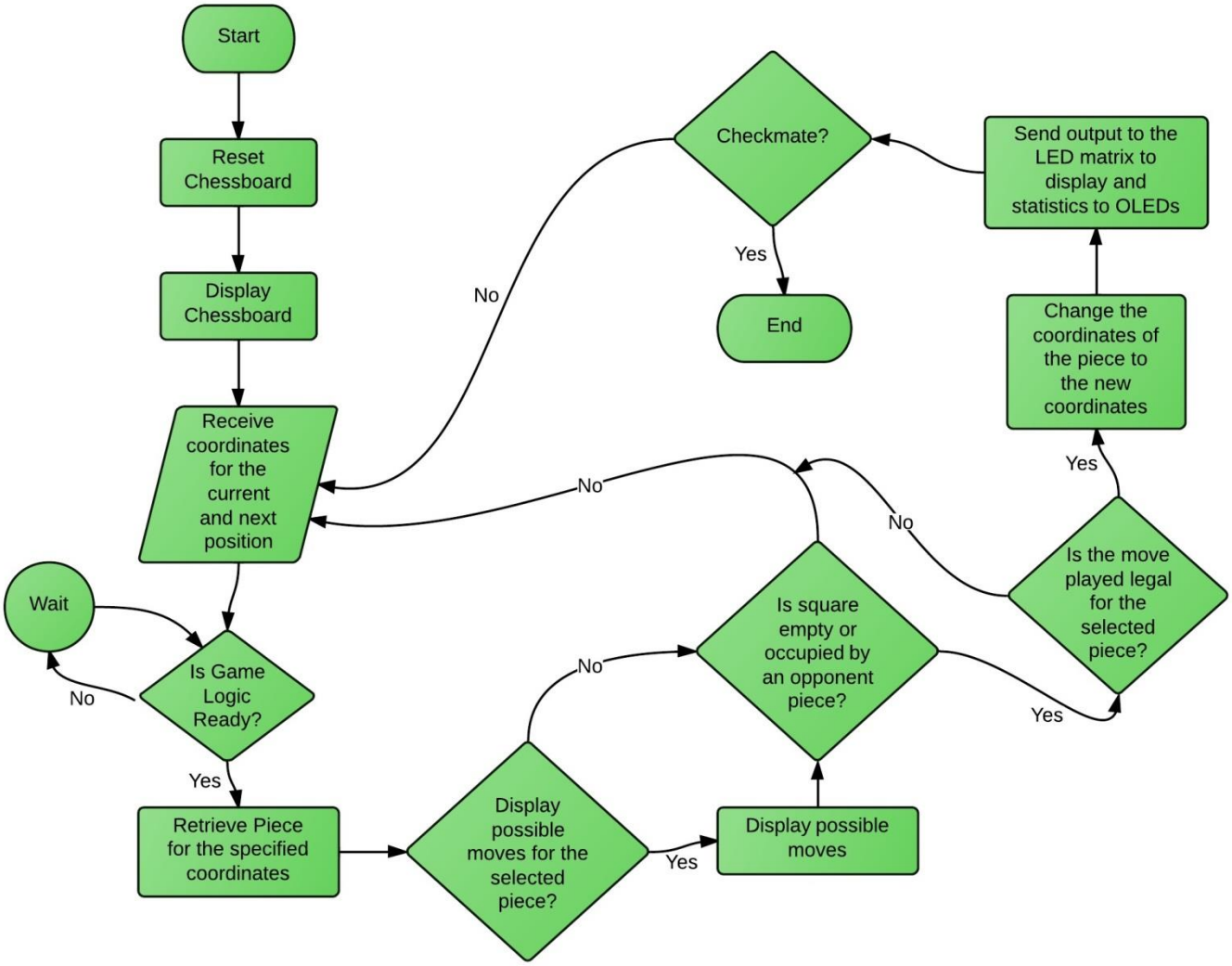


Figure 1: Main program flow. The same steps are repeated for both the players after resetting the chess board.

Appendix B: Hierarchical Block Diagram of Code Organization

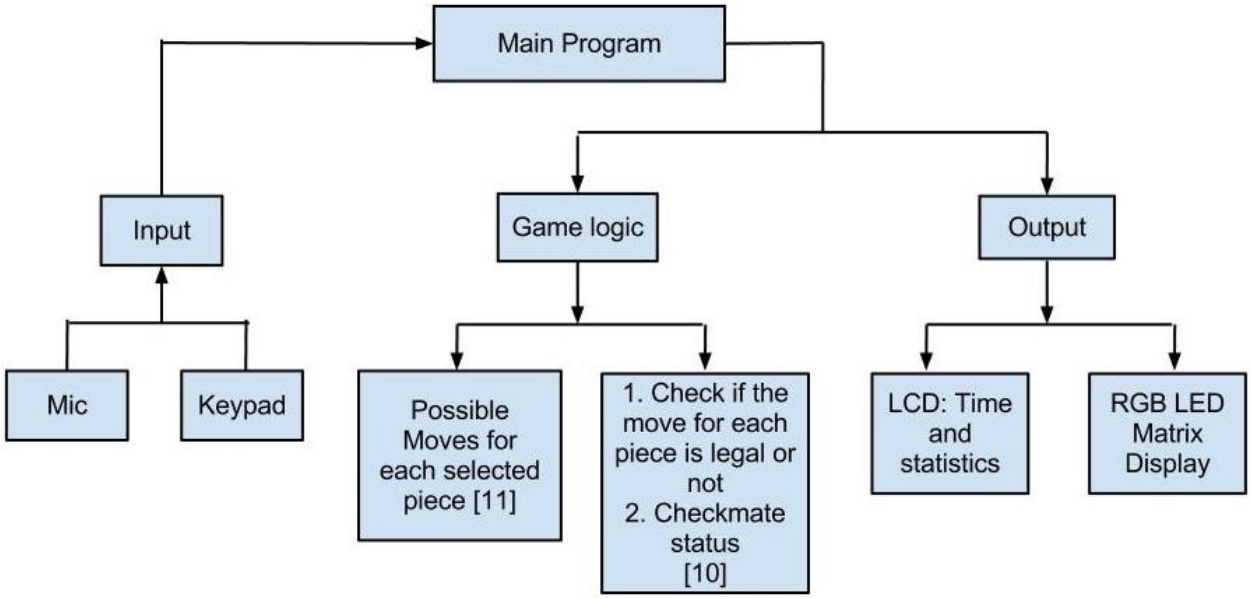
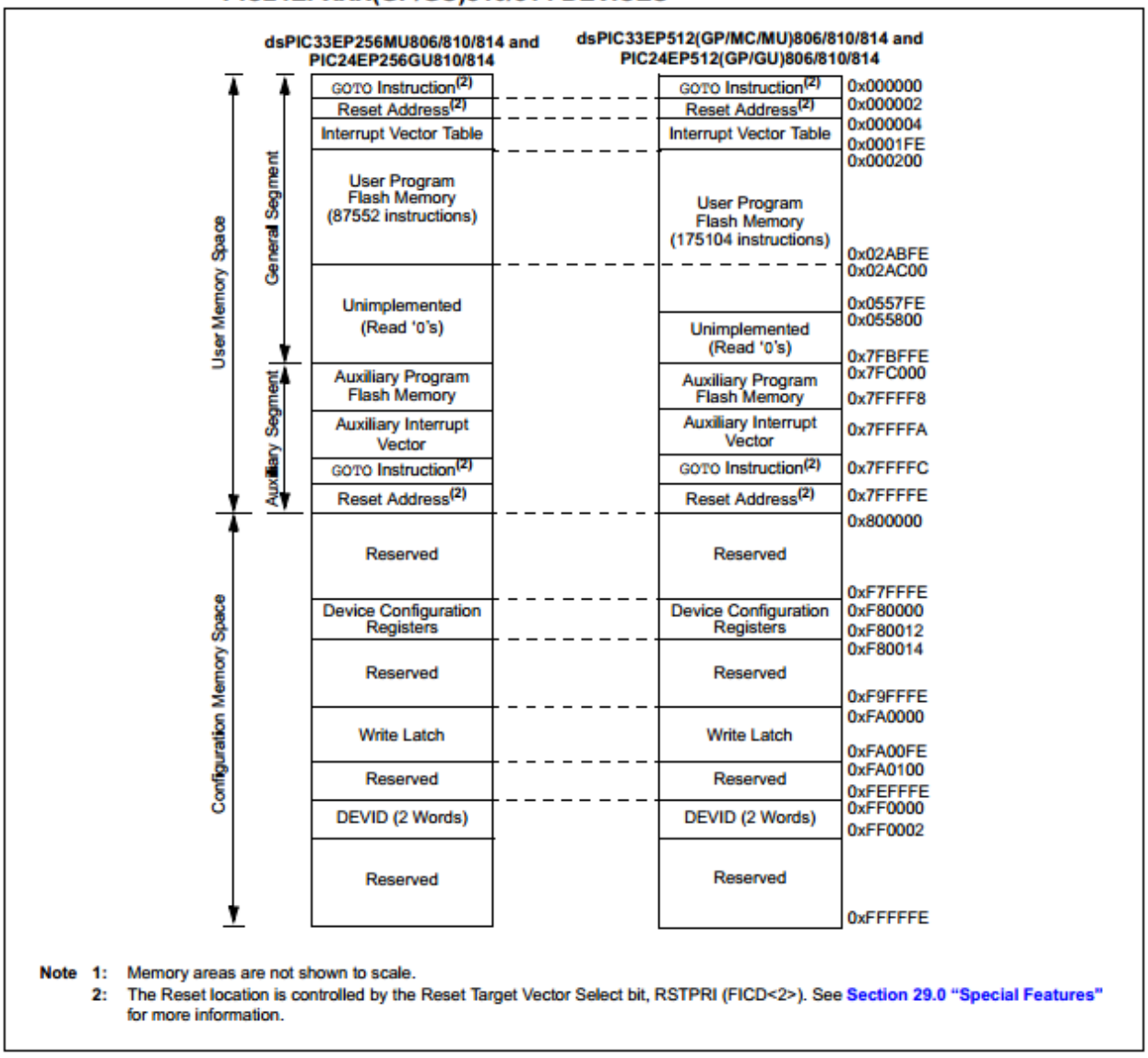


Figure 2: Hierarchical Block Diagram

Appendix C: Related Images

FIGURE 4-1: PROGRAM MEMORY MAP FOR dsPIC33EPXXX(GP/MC/MU)806/810/814 and PIC24EPXXX(GP/GU)810/814 DEVICES(1)



Note 1: Memory areas are not shown to scale.
2: The Reset location is controlled by the Reset Target Vector Select bit, RSTPRI (FICD<2>). See **Section 29.0 "Special Features"** for more information.

Figure 3: Memory Organization for PIC24EP512GU810 [13]

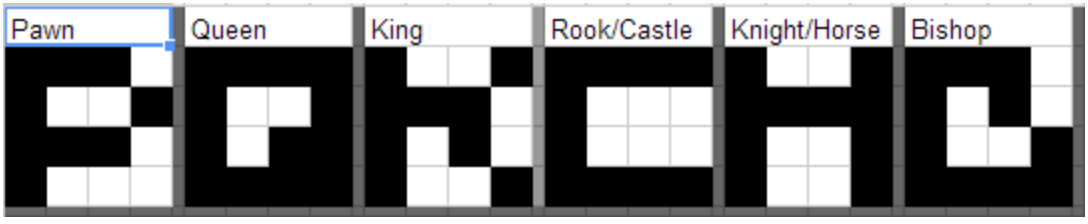


Figure 4: Characters display on the RGB LED matrix (4x4)

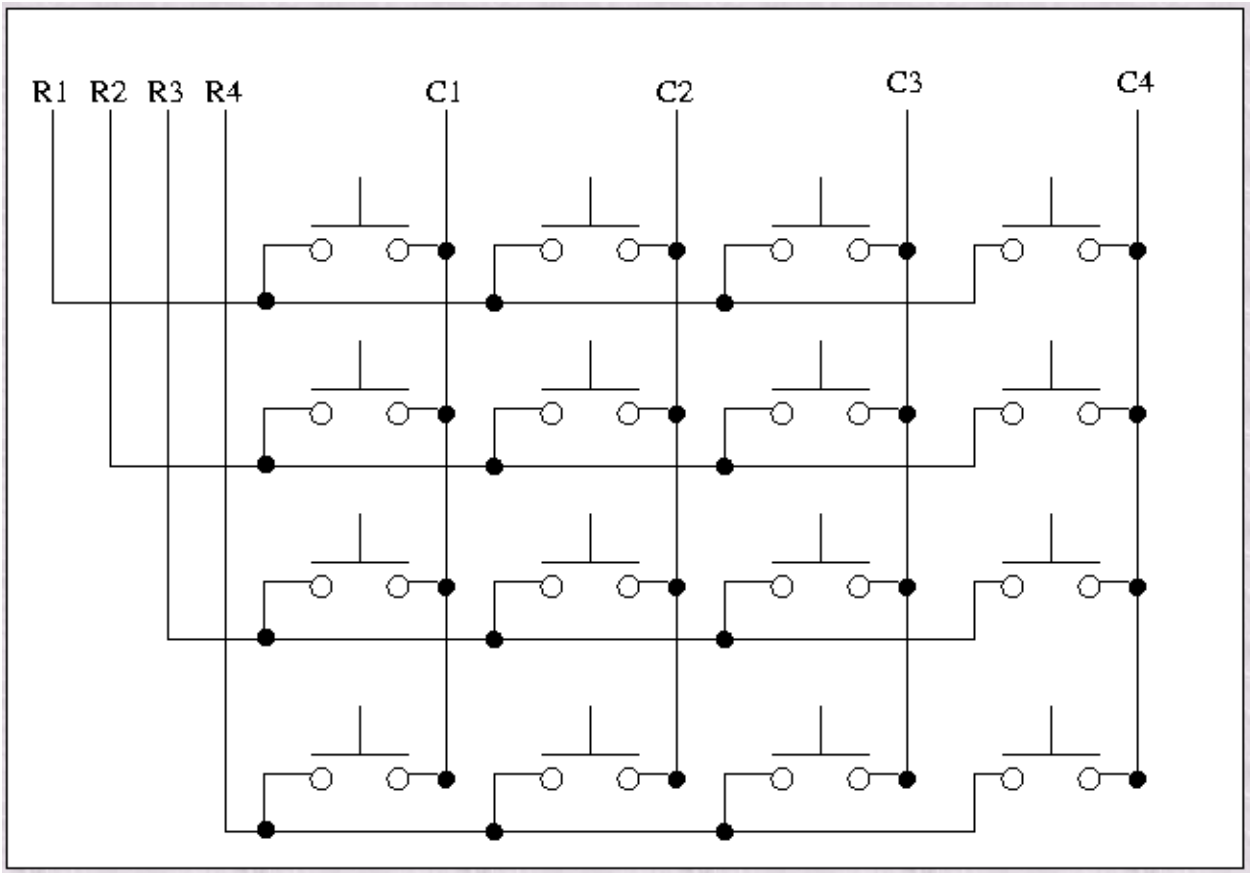


Figure 5: Connections for Keypad Scanning [14]

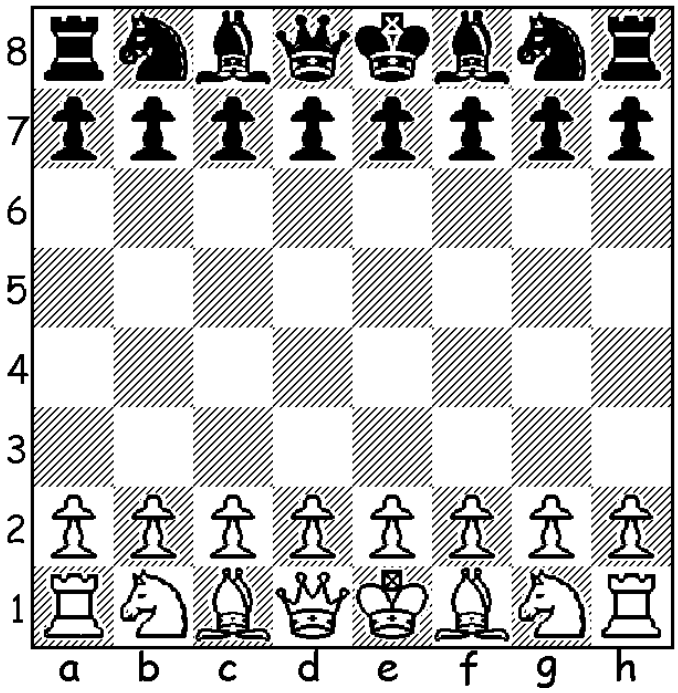


Figure 6: Chessboard Layout [15]