EE 443 Project Report: Implementation of Two-Player Chess With Timer on SoC Board

Eren Mermer, Sait Sevban Cander, Uğur Tekinalp

I. Summary

In this project, a standard 2-player chess game with time mode is implemented on the Altera DE1-SoC board. The game can be played using a PS/2 mouse. The game board is displayed on the screen with 640x480 resolution. The timing options can be selected using the switches on the DE1-SoC board and the remaining time is shown on the seven-segment displays of the board.

II. Introduction and Broader Impact

Chess, with its 1500-year history, is one of the oldest and most popular games in the world. Played by two opponents on a checkered board with specially designed pieces of contrasting colors, commonly white and black[1]. Throughout history, chess is played using physical pieces. Now, with the help of this project, chess can be played virtually. There will be no need to take the chess set everywhere.

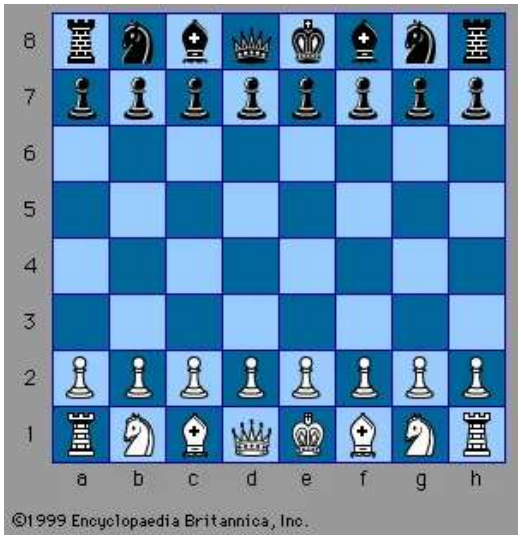

Fig.1 A representational chess board[1]

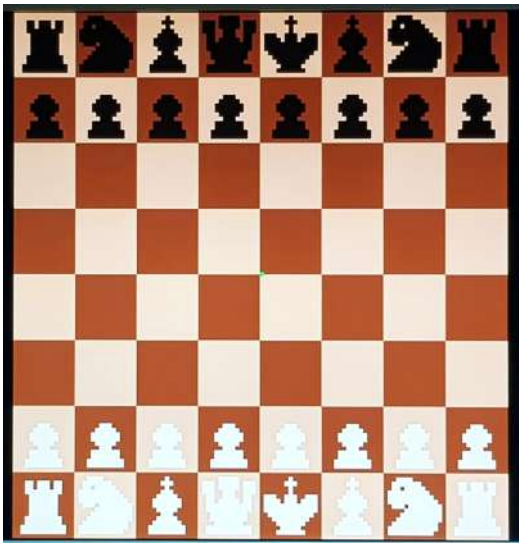

Fig.2 Chess board that is printed to the screen in this project

First of all, there are an enormous number of possibilities in chess; more than the number of atoms in the universe. “Each possible move represents a different game, a different universe in which you make a better move. By the second move, there are 72084 possible games. By the third, 9 million. By the fourth…which means that the first move can be terrifying...”[2]. This makes our job harder. There are many rules to define and these rules should be defined carefully so that they cover all of the possibilities. In this project, the main rules are defined.

Secondly, one of the key challenges of this project was getting data from the PS/2 mouse. Because of the fact that the mouse sends the 3-byte data packet without identification, it was very challenging to decode the incoming data from the mouse since the PS/2 mouse sends the data randomly over time. To overcome this, we decoded the data by assuming that the bit-3 of the first byte in the data packet was always equal to 1. Then, the other bytes are saved respectively after that byte. Consequently, the alignment error was fixed.

Another challenge was to implement the specially designed chess pieces to the virtual environment. In this project, we had a limited number of pixels in this but on the other hand we needed the appropriate symbols for the pieces of chess. There were no images that were compatible with our resolution on the internet, so we drew all the shapes of pieces by hand, pixel by pixel. We used a website[3] to draw the symbols with 30x30 resolution. After that, we converted these symbols into arrays to represent the shape so that we could use the arrays to print the designed shapes on the screen.

In the future, a chess engine can be added to the project. This engine may include the chess openings and have a feature of calculating up to 24 moves ahead. This may require a high CPU power but the DE1-SoC board has ARM Cortex-A9 processors so it may handle it. Since it can calculate all possible moves, it can also show which player has the positional advantage or piece advantage over a scale. With this implementation, players can play with the engine. Also if the internet were to be used, people now would be able to play chess online using this project.

III. Task Description and Design Implementation

We used different inputs and outputs for the implementation. These individual implementations are explained in the sub-parts below. The code was written in C since it is too complex to be coded in Assembly. Both interrupt and polling methods are used in the code. The interrupt method is used for obtaining data from the mouse and the polling method is used to obtain data from the timer and the pushbuttons. The game does not require additional hardware that would not be available to an average computer user or consume additional power since it requires a mouse, a monitor and a speaker. Every computer user has a monitor and PS/2 mice, speakers are mostly cheap hardware and these hardware and the board do not consume much power thus cost or power is not an issue. Other than the loading sequence, the game is relatively fast and the moves are displayed smoothly. For the market, online chess platforms are competitors to this project.

A. Implementing the Graphics

The first task of this project was to print the chess board to the screen without the pieces. Before starting that, some pixel calculations are done. The pixel buffer provides an image resolution of 320x240 pixels. Since a square-shaped chess board is implemented to the screen and it is indented to the middle, only 240x240 pixels are used. There are 64 squares on the chess board, so every square has 30x30 pixel resolution before replication. So, we defined an array that keeps the data of pixels of the chess board. This array changes its color data every 30 pixels both in horizontal or vertical. Then the data inside this array is printed to the screen. This resulted in a screen as shown in Fig.3. The program prints this board once immediately after the game starts.

Fig.3 Chess board without pieces

After printing the empty board, the second task is to print the chess pieces on the screen. This means the pixel data of 30x30 chess piece images are needed. The images we wanted were not available on the internet, so we drew it ourselves. First, we found a free online pixel art drawing tool[3] and then we drew all of the pieces using this tool. An example of these drawings is shown in Fig.3. After drawing the pieces, we transformed those drawings into 30x30 arrays. An example of these arrays is shown in Fig.4. The arrays are included in the code as a separate txt file. In the array, the elements representing the pixels of the piece are valued as zero.

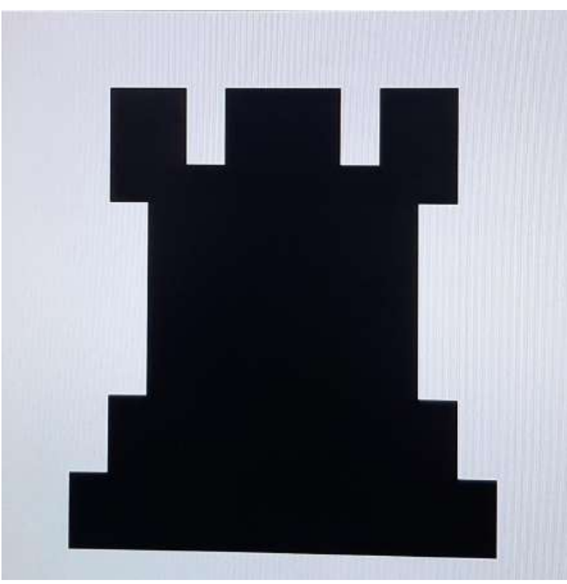

Fig.3 The drawing of the rook

```
int rook[30][30] = {
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1},
    {1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1},
    {1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1},
    {1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1},
    {1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};
```

Fig.4 The pixel data of the rook

After getting the pixel data of each piece, another function that prints these data to the screen is needed. We created another function that takes the most left and the top pixel of each square as a reference, then prints the desired piece with its corresponding color to the desired square of the chess board.

B. Using the PS/2 Mouse

To implement a mouse to this project, the PS2 port of the DE1-SoC board is used. PS/2 mice send the data in 3-byte packets. To get the data in the first place, the initialization command(0xF4) is written to the base address of the PS2 port. Then, to get interrupts from the mouse, the interrupts are enabled by sending 0x11 to the control register of the

mouse according to the manual[4]. At the beginning of the program, interrupt services and the GIC (General Interrupt Controller) are initialized in C programming. Then, every time the mouse sends the data, the program goes to the interrupt subroutine and checks for the ID of the interrupt. The ID of the PS2 port is 79. So, if the interruption ID equals 79, the program goes to the next step which is handling the interrupt.

When the program knows that the data is incoming with an interrupt, it should decode and save the data. In this project, first the data is decoded by using the fact that the bit-3 of the first byte is always logic 1. However this information is not enough because the bit-3 of the second and third bye can also be logic 1. Since we are not using the middle button and the left button of the mouse, those bits will always be logic 0. So 3 bits of the first byte is known. This fact is used to differentiate the first byte and then to save the second and third byte respectively.

Lastly, a cursor is shown on the screen according to the horizontal and vertical coordinates of the mouse. The color of the cursor is bright green in order to follow the cursor easily. Every time the mouse moves, the change in the coordinates of the mouse are added to the coordinates of the cursor in the interrupt subroutine. After adding the coordinates, the cursor is printed to the screen again. When the left button of the mouse is pressed, the program saves the present coordinates of the cursor.

C. Calculating and Displaying Time and Ending the Turn

In this project, there are also time options and players can choose in which time range they want to play. They can choose the increment of the time (in seconds) for each move and the total time (in minutes) for each player by using switches according to Fig.5. To play with increments, players should turn on one of the switches from the increment part (last 3 switches, bits 7 to 9 of the switch address), and one of the switches from the total time part.

If none of the increment switches are turned on, the time mode will nove have increments i.e. the time will not increase after making a move. If none of the time switches are turned on, the game will not have a time mode. For example, if the player wants to play in 5+3 mode, switch 2 and switch 8 should be turned on. Note that switch 0 is the rightmost switch and switch 9 is the leftmost switch.

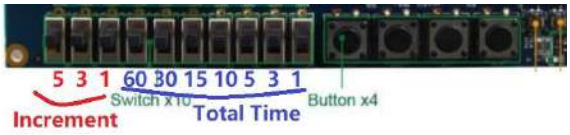

Fig.5 Corresponding data for each switch

In order to implement the timer to the program, the FPGA interval timer is used. First, the FPGA timer is configured for counting one second. The FPGA timer raises its interrupt flag when one second is completed. The program lowers the interrupt flag and decreases one second from the time of the player playing the current turn. The remaining time for each player is displayed on the seven-segment displays on the DE1-SoC board.

Each player should press any of the pushbuttons shown in Fig.5 after making a move. When the player presses a pushbutton, their counter will stop and then the remaining time of the opponent will start counting down.

D. The Game Algorithm

In this project, a 8x8 two dimensional integer array is used to represent the chess board. Each piece has its special number in the array. To store whether the piece is black or white, another 8x8 two dimensional integer array is used. Every time a move is made by a player with the mouse, these two arrays are changed accordingly and the pieces on the screen are drawn again. Also after each move, when a player presses one of the pushbuttons to end his/her turn, the board is flipped so that the other player does not have to play from a reversed point of view. To do that, the two dimensional array that contains the data of the pieces is changed into its symmetric version with respect to the origin. This way, both players can see the chess board from their own perspective.

There is another 8x8 two dimensional integer array which stores the possible moves of the selected piece. The algorithm calculates the possible moves of the selected piece and writes the data to this array. Then the array is used to calculate if the move is valid or not. If the move is valid, the program performs the move and displays it on the screen. The game is over when one of the players runs out of time or the player's king is directly attacked by one of the pieces of the opponent and has no possible move to escape the check e.g. is checkmated.

E. Audio

The program also uses the audio out port of the codec. The task for this part is to play a sound effect

every time that a player makes a valid move. Firstly, we searched for license-free sounds on the internet. After finding and downloading an appropriate sound that lasts approximately half a second,[5] we converted the sound data into a one dimensional array using the MATLAB function “audioread”. The function uses 44.1kHz sample rate, which is compatible with Codec on the DE1-SoC board. Then the array is saved in a file called ‘sfx.txt’. The main function of the program includes that file and sends the sound data to the audio out port when a player makes a valid move. To hear the sound, players should connect a speaker or a headset to the audio out port of the DE1-SoC board.

References

[1] “Chess,” *Encyclopædia Britannica*. [Online]. Available: https://www.britannica.com/topic/chess. [Accessed: 18-Jan-2022].

[2] Person of Interest, “Harold Finch VS. Machine (Chess)”, *Youtube*, May. 12, 2015 [Video Recording]. Available: https://www.youtube.com/watch?v=iz5EFsSbQ7U

[3] “Draw pixel art online - 30X30,” *Pixilart*. [Online]. Available: https://www.pixilart.com/draw/30x30-6268716eed3fa70. [Accessed: 18-Jan-2022].

[4] “Altera University Program DE1-SoC Computer Manual.” [Online]. Available: https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/Computer_Systems/DE1-SoC/DE1-SoC_Computer_ARM.pdf. [Accessed: 18-Jan-2022].

[5] “Move.mp3” June. 12, 2021 Available: https://github.com/ornicar/lila/tree/master/public/sound/lisp [Accessed: 18-Jan-2022].

License notice:

“Copying Lila” *Github.* https://github.com/ornicar/lila/blob/master/COPYING.md

By EdinburghCollective: http://lichess.org/@/EdinburghCollective

The sampled and adapted mp3 file was downloaded from the link. The file is under CC BY-NC-SA 4.0 license. The file was indicated as free use in the github page. The use is noncommercial and is under fair use.